Nonlinear Trajectory Optimization with Path Constraints Applied to Spacecraft Reconfiguration Maneuvers

by

Ian Miguel García

S.B. Aeronautics and Astronautics, Massachusetts Institute of Technology, 2003
S.B. Mathematics, Massachusetts Institute of Technology, 2003

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author .....
Department of Aeronautics and Astronautics
May 20, 2005

Certified by .....
Jonathan P. How
Associate Professor
Thesis Supervisor

Accepted by .....
Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Nonlinear Trajectory Optimization with Path Constraints Applied to Spacecraft Reconfiguration Maneuvers

by

Ian Miguel García

Submitted to the Department of Aeronautics and Astronautics
on May 20, 2005, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Future space assembly and science missions (e.g., Terrestrial Planet Finder) will typically require complex reconfiguration maneuvers involving well coordinated 6 DOF motions of the entire fleet of spacecraft. The motions must also satisfy constraints such as collision avoidance and pointing restrictions on some of the instruments. This problem is particularly difficult due to the nonlinearity of the attitude dynamics and the non-convexity of some of the constraints. The coupling of the positions and attitudes of the N spacecraft by some of the constraints adds a significant complication because it requires that the trajectory optimization be solved as a single $6N$ DOF problem.

This thesis presents a method to solve this problem by first concentrating on finding a feasible solution, then developing improvements to it. The first step is posed as a path planning problem without differential constraints and solved using Rapidly-exploring Random Trees (RRT's). The improvement step is posed as a feasible nonlinear optimization problem and solved by an iterative optimization similar to a sequential linear programming method. The primary contribution of the thesis is an improvement to the basic RRT algorithm that replaces the connection step with a more complex function that takes into account local information about the constraints. Two functions are proposed, one based on artificial potential functions, and another based on a random search. The results show that the new RRT with either of these connection functions is faster and more reliable for a difficult sample problem. The combination of an RRT with the new connection functions, and the improvement step, is also demonstrated on several challenging spacecraft reconfiguration problems with up to 16 spacecraft (96 DOF). The solution technique is shown to be robust and with computation times fast enough to embed in a real-time optimization algorithm as part of an autonomous onboard control system.

Thesis Supervisor: Jonathan P. How
Title: Associate Professor

Acknowledgments

I would like to thank my advisor, Prof. Jonathan How, for sharing with me the questions and insight that made this work possible, and for going out of his way to help me complete it. I want to express my gratitude as well to the members of the research group who, I have to honestly say, are a bunch of funny, smart and helpful people with whom working has been real pleasure.
I would also like to thank the MIT Department of Aeronautics and Astronautics for awarding me with a Departmental Fellowship which helped support part of this work.

To Papá, Mami and Javi

This work was funded under NASA Grant NAG5-10440 and Cooperative Agreement NCC5-729 through the NASA GSFC Formation Flying NASA Research Announcement. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Nonlinear Trajectory Optimization

Robots moving in a factory floor, airliners flying across the country, unmanned aircraft flying over a battlefield, rockets soaring to space, and spacecraft cruising toward distant planets of the Solar system, will all move following a planned trajectory. Trajectory design consists of planning the path that a vehicle can follow. This path must be consistent with the dynamics of the vehicle, such as having a maximum speed, or a minimum turning radius. Sometimes these trajectories are also required to satisfy other physical constraints, like not going through walls or not colliding with other vehicles. In many cases the trajectories should make optimal use of fuel, energy, or time.

Many methods have been developed to solve trajectory optimization problems [3]. A large fraction of these methods is based on formulating and solving a constrained nonlinear optimal control problem using indirect or direct methods. Indirect methods use the solution to Pontryagin's maximum principle to transform the optimal control problem into a two-point boundary value problem [39, 50]. This TPBV problem can then be solved by numerical methods. Direct methods transform the continuous optimal control problem into an equivalent nonlinear programming problem (NLP) by discretizing the trajectory at certain points and describing the constraints and cost function at these points [3, 16]. The solution to this second problem can then be found

with a nonlinear optimization solver such as NPSOL, SNOPT or CFSQP [13, 14, 26]. There are advantages and disadvantages of both indirect and direct methods, but direct methods have better convergence properties and thus can perform well with a poor initial guess. Many direct methods have been developed based on different ways to transform the original control problem into an optimization problem [3]. Two direct methods developed recently are the pseudospectral Legendre method [8, 44], and a method based on parameterizing *differentially flat systems* with B-spline curves [32, 37, 33]. The pseudospectral Legendre method has been implemented in the DIDO software package by Fahroo and Ross [46, 44], while the B-splines parametrization method, better known as the *nonlinear trajectory generation* (NTG) algorithm, has been implemented in the NTG library from Caltech [33].

Another set of methods that are also used to solve trajectory optimization problems are based on geometric considerations. These method were originally developed to solve robot path planning problems [41, 23, 25]. Recent developments in this field include randomized path planning algorithms, among which Randomized Roadmaps and Rapidly-exploring Random Trees (RRTs) are the best known [19, 24].

In general, however, nonlinear trajectory optimization problems are very difficult, regardless of what type of method is used to solve them. The difficulty is due to the nonlinear nature of the problems and the time complexity of the numerical methods that must be used. Even without regard for optimality, just finding a feasible solution for the general path planning problem with general constraints is NP-hard [41, 5]. This means that, in general, the time complexity of a path planning problem is believed to be exponential in its number of degrees of freedom, even when optimality is not required. A problem with this time complexity is very hard to solve numerically, and even small cases with few degrees of freedom cannot be solved in a practical amount of time. This difficulty is well known since the origins of optimal control theory, and it was what Richard Bellman referred to as the "curse of dimensionality" [2]. The randomized path planning algorithms mentioned before alleviate this difficulty by relaxing the guarantees of finding a solution, and as a result, they have been able to solve larger problems. A good approach then to solve large problems is to use a

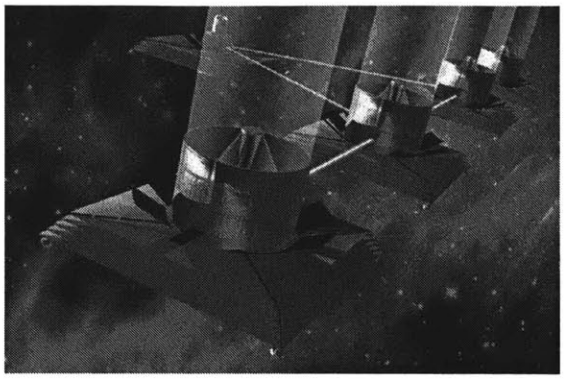


Figure 1.1: The Terrestrial Planet Finder mission requires precisely coordinated motions of several spacecraft [28, 36]

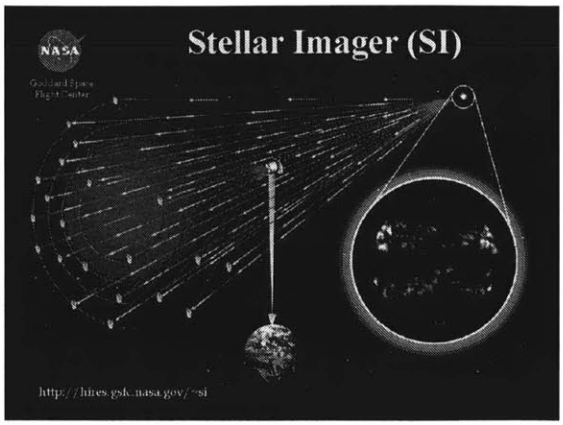


Figure 1.2: The proposed Stellar Imager will have a formation of 20 to 30 satellites [6, 35]

randomized algorithm to find a feasible solution, and then a nonlinear optimal control algorithm to improve it.

Another difficulty in some trajectory optimization problems is that they may have discontinuous dynamics, constraints or cost function. These discontinuities consist of sharp changes at different points of the trajectory, and are avoided by most existing solution methods, which are limited to continuous problems. However, certain type of discontinuous problems can be transformed into continuous equivalents which can then be solved with existing methods.

This thesis presents methods to solve both problems, the hard nonlinear trajectory optimization problem with many degrees of freedom, and the trajectory optimization with discontinuous constraints. The method to solve large problems is motivated by a trajectory optimization problem involving a spacecraft formation. This problem is interesting, useful, and has all the difficult characteristics that make it an ideal benchmark for the method developed here.

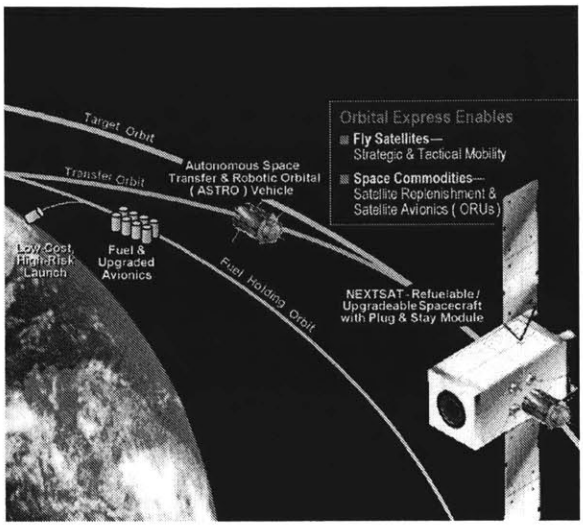


Figure 1.3: DARPA Orbital Express requires coordinated maneuvers of several spacecraft [49]

1.2 Spacecraft Formation Reconfiguration with Position and Attitude Constraints

Formation flying of multiple spacecraft is an attractive technology for many planned space science such as the Terrestrial Planet Finder and the Stellar Imager shown in Figures 1.1 and 1.2, as well as autonomous space assembly missions like the DARPA Orbital Express shown in Figure 1.3 [29, 1, 28, 34]. These missions will typically require complex initialization maneuvers and/or formation reconfiguration maneuvers involving well coordinated motions of several spacecraft. These maneuvers consist of moving and rotating a group of N spacecraft from an initial configuration to a desired target configuration, while satisfying an arbitrary number of constraints (e.g., see Figure 1.4). These constraints may consist of collision avoidance, restrictions on the region of the sky where certain spacecraft instruments can point (e.g., a sensitive instrument that cannot point at the Sun), or restrictions on pointing toward other spacecraft (e.g., requirements on maintaining inter-spacecraft communication links and having cold science instruments avoid high temperature components on other vehicles). It is also desirable to perform these maneuvers with the lowest possible fuel

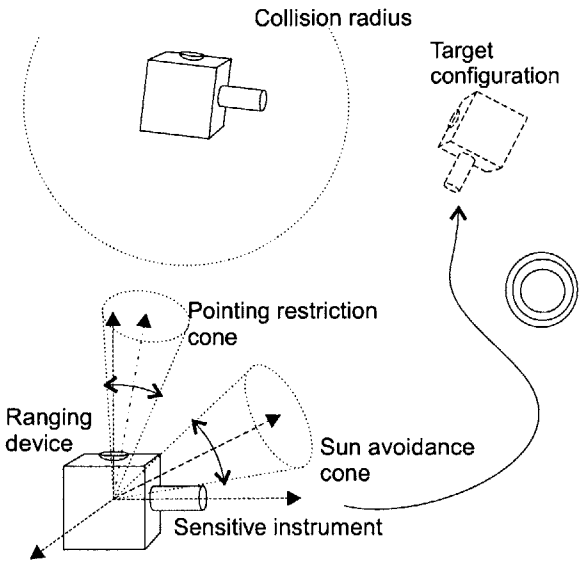


Figure 1.4: The formation reconfiguration problem

expenditure.

This problem is particularly difficult due to the nonlinearity of the attitude dynamics, the non-convexity of some of the constraints, and the coupling between the positions and attitudes of all spacecraft. The nonlinearity of the spacecraft attitude dynamics makes the attitude trajectory design problem difficult in itself [18]. Although numerous solutions exist for the attitude control problem alone, its nonlinearity adds considerable difficulty to the general spacecraft reconfiguration problem. The non-convex constraints place this problem in a general class of path planning problems with obstacles that were described in the previous section as computationally hard and therefore with a computational complexity exponential in the number of degrees of freedom. Finally, some of the pointing constraints introduce coupling between the positions and attitudes of the entire fleet. As a result, the trajectory design must be solved as a single $6N$ DOF problem instead of N separate 6 DOF problems. Spacecraft trajectory optimization has been the subject of extensive research and many methods have been developed to plan translation maneuvers, attitude maneuvers, and in some cases for the combinations of both types of maneuvers. An overview of this research and how it relates to the work in this thesis is presented in Chapter 2.

1.3 Solution Approach to Hard Trajectory Optimization Problems

The approach chosen to tackle the computational difficulty of nonlinear trajectory optimization problems with path constraints is to concentrate first on finding any feasible trajectory. When a feasible trajectory is found, then a second stage focuses on improving the solution while maintaining feasibility. These two parts are called the *path planner* and the *smoother* in this thesis. This separation is a natural approach that has been used many times in difficult path planning problems, including in spacecraft trajectory design [38].

The problem of finding a feasible path between two points with an arbitrary number of constraints is a classical path planning problem that has been studied for many years [41, 5, 23]. The difficulty of this problem makes it intractable to find a solution with guarantees for problems with more than a few (e.g., 3–6) dimensions. However, by relaxing the guaranteed completion, randomized path planning algorithms such as the Probabilistic Roadmaps (PRM) have been shown to solve larger problems [19]. These algorithms and its more recent derivatives have solved problems of more than 20 dimensions [19, 47, 7, 24, 38]. Rapidly-exploring Random Trees (RRT), a more recent variant of these randomized planners, is used in this thesis because it performed better than other algorithms in our experimental comparisons [24].

There has been work on applying PRMs and RRTs to spacecraft trajectory design [24, 10, 11, 38]. This body of work includes differential constraints, which in the case of spacecraft means that the trajectories must be consistent with the kinematics and dynamics of spacecraft. In general, these constraints limit the expansion of the randomized search trees and increase the solution time. In the *planner*, the differential constraints have been removed. Instead, the direct trajectories between two points consist of rest-to-rest straight-line translations and eigen-axis rotations at constant rates. It is possible to do this because the spacecraft is at rest at each node of the search tree, and the links between nodes are feasible trajectories. The trajectory generated by the algorithm consists of a sequence of states, connected by

these feasible "direct trajectories". This trajectory is then passed to the *smoother*.

The *smoother* improves the cost of the trajectory found previously. This trajectory is represented by the states and control inputs sampled at fixed time-steps. The smoothing problem is posed as a nonlinear optimization with nonlinear constraints. The cost function, dynamics and constraints are then linearized, and the resulting equations are solved as a linear program. These steps are repeated iteratively until the cost associated with the trajectory cannot be improved.

After each iteration, the solution may violate some of the constraints by a small amount, so there is an additional step to recover a feasible solution. This step uses knowledge specific to the trajectory design problem, and is the main difference between the smoother and a similar algorithm, the Sequential Linear Programming method (SLP) [9]. The recovery step is discussed further in Section 2.2.2.

This algorithm is also different from other trajectory smoothing algorithms proposed in the literature because it improves the cost of the trajectory as a whole, it is deterministic, and always remains feasible [38, 51].

1.4 An Improved Planner

This thesis also presents an extension of the planner that decreases the solution times by approximately two orders of magnitude, thus enabling the solution of problems with up to sixteen spacecraft. The extension consists of replacing the function that generates a simple trajectory between two points in the RRT, by a function that is more complex, but also more likely to find the connection. Most randomized planners use a simple function that generates a straight line or a simple rotation that stops after reaching the second point or hitting an obstacle, with the intention of generating as many connections and growing the random trees as fast as possible. However, there is also value in spending more time choosing better connections and thus generating a more efficient search tree. This trade-off between complexity and effectiveness appears in many problems (e.g., other path planning literature [25], and basic optimization, such as first-order steepest descent versus the second order Newton optimization

methods).

Two new connection approaches are proposed and evaluated. The first connection approach is based in a local randomized search. The second approach is similar to gradient descent on artificial potential functions [20]. Since the obstacles in the spacecraft reconfiguration problem are constraints that can be written in the form $g(x) \leq 0$, the gradient descent is posed as a nonlinear optimization problem and solved with a feasible solver. The intermediate solutions of the solver are then used as the trajectory connecting the points. The results show that both extensions significantly improve the solution times of the basic *planner*.

1.5 Trajectory Optimization with Discontinuous Constraints or Cost

The spacecraft reconfiguration problem is a good example of a trajectory optimization problem with continuous path constraints that must hold for every point of the trajectory, such as the collision avoidance and the pointing constraints. However, there are trajectory optimization problems with constraints that must hold only in certain regions of the space, in certain section of the trajectory, or during a time interval. In other cases the constraints may hold over the whole trajectory, but change sharply in some sections. These sharp changes may exist also in the cost of the trajectory, for example in travel time of a vehicle that crosses from traveling over land to traveling over swamp. In all the cases mentioned before, the problems have discontinuities in the constraints or in the cost function.

Typical examples of problems with discontinuities have integral constraints, or have a cost function that changes sharply depending on the area the trajectory goes over. These examples are simple, but also general enough that can be used as a basis for other more useful problems such as minimum risk trajectory design. Fig. 1.5 shows a UAV path planning problem in which the objective is to minimize the total risk of failure. The best trajectory is not the shortest path but the one that has the shortest

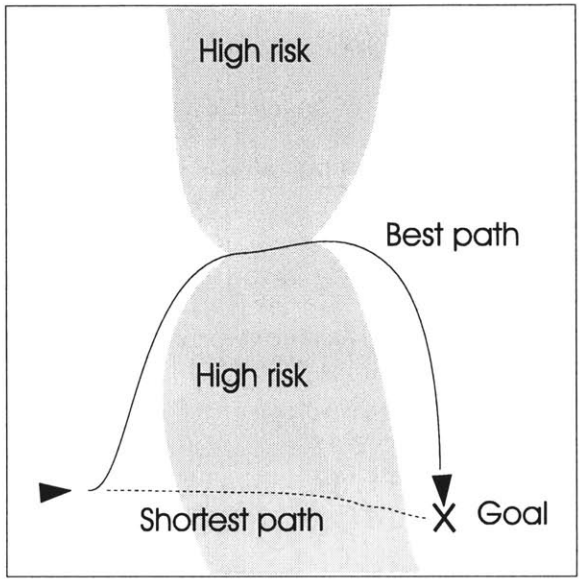


Figure 1.5: The best trajectory minimizes the length over the high risk area

length over risky terrain [21]. This problem is discontinuous because the penalty on the path length changes sharply when the aircraft enters the high risk region.

Most of the recent trajectory optimization methods assume some degree of continuity in the problems and are not well suited to handle discontinuous constraints. However, it is possible to modify certain types of problems with discontinuities so that they can be formulated as continuous problems that can be solved by current nonlinear optimal control methods. This thesis considers a formulation of the discontinuous problem that can be transformed into a continuous formulation, and shows how this transformation can be done.

1.6 Overview

Chapter 2 describes the spacecraft reconfiguration problem and the method to solve this problem based on separation. This chapter also describes the planning and smoothing steps that are characteristics of the separation procedure, in particular the planner and smoother tailored to the reconfiguration problem. The planner and the smoother are then compared against DIDO, a recent nonlinear optimal solver.

Chapter 3 describes the improvement to the RRT planner based on two new connection methods that use local information. The improvement is illustrated with several examples of reconfigurations of spacecraft formations of up to 16 spacecraft.

Chapter 4 describes a class of discontinuous problems that can be transformed into continuous problems that can be solved by the NTG and DIDO optimal control software packages, and the method to do this transformation. The method is illustrated with two examples of discontinuous problems: the sensor problem and the weighted shortest path problem.

Chapter 2

Trajectory Optimization for Satellite Reconfiguration Maneuvers

The spacecraft trajectory design problem for the unconstrained translation and attitude maneuvers has been the subject of extensive research with successful results. In contrast, the trajectory design problem with constraints is more difficult and has attracted attention more recently. Most of the solutions for this problem find either the translation or the attitude trajectories. Some of these solutions include potential functions methods [20, 31], geometric/heuristic methods [15], Mixed-Integer Linear Programming [42], and randomized algorithms [11, 10]. The trajectory design problem with combined translation and attitude has also been investigated. Some recent solutions are based on geometric/heuristic methods [48] and randomized algorithms [38]. However, the geometric and heuristic methods are problem specific and cannot be extended to solve the general reconfiguration problem. The randomized algorithms were used to solve a problem with a single spacecraft and no pointing constraints. The reconfiguration problem addressed in this chapter is more general and on a larger scale than the problems considered before. For example, we design maneuvers for 4 spacecraft with 24 DOF. The following sections outline the solution technique developed to solve these problems, followed by several demonstrations.

2.1 The Spacecraft Reconfiguration Problem Formulation

The general reconfiguration problem consists of finding a trajectory, represented by the state and control inputs of N spacecraft, from time 0 to T. This state consists of

$$\boldsymbol{r}_i(t) \in \mathbb{R}^3, \quad \dot{\boldsymbol{r}}_i(t) \in \mathbb{R}^3 \tag{2.1}$$

$$\boldsymbol{q}_i(t) \in \mathbb{SO}^3, \quad \boldsymbol{w}_i(t) \in \mathbb{R}^3 \tag{2.2}$$

where $i \in 1 \ldots N$ indicates the spacecraft. $\boldsymbol{r}_i(t)$ is the position of its center, $\dot{\boldsymbol{r}}_i(t)$ its velocity, and $\boldsymbol{w}_i(t)$ its angular velocity, all measured with respect to a local inertially fixed frame. The attitude quaternion is $\boldsymbol{q}_i(t)$. The input consists of

$$\boldsymbol{T}_i(t) \in \mathbb{R}^3 \text{ and } \boldsymbol{M}_i(t) \in \mathbb{R}^3 \tag{2.3}$$

where $\boldsymbol{T}_i(t)$ are the forces and $\boldsymbol{M}_i(t)$ the moments. Let $\boldsymbol{x}_i(t)$ be a point of the trajectory of a single spacecraft at time t,

$$\boldsymbol{x}_i(t) = [\boldsymbol{r}_i(t), \dot{\boldsymbol{r}}_i(t), \boldsymbol{T}_i(t), \boldsymbol{q}_i(t), \boldsymbol{w}_i(t), \boldsymbol{M}_i(t)], \tag{2.4}$$

for $i \in 1 \ldots N$, and

$$\boldsymbol{x}(t) = [\cdots, \boldsymbol{x}_i(t), \cdots] \tag{2.5}$$

a point in the composite trajectories of all the spacecraft at time t.

For the deep space missions considered, the translational dynamics are approximated with a double integrator

$$\begin{bmatrix} \dot{\boldsymbol{r}}_i(t) \\ \ddot{\boldsymbol{r}}_i(t) \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_i(t) \\ \dot{\boldsymbol{r}}_i(t) \end{bmatrix} + \begin{bmatrix} 0 \\ I \end{bmatrix} \boldsymbol{T}_i(t) \tag{2.6}$$

and the attitude dynamics, in quaternion notation, are

$$\dot{\boldsymbol{q}}_i(t) = \frac{1}{2}\Omega_i(t)\boldsymbol{q}_i(t) \tag{2.7}$$

$$J\dot{\boldsymbol{w}}_i(t) = -\boldsymbol{w}_i(t) \times (J\boldsymbol{w}_i(t)) + \boldsymbol{M}_i(t) \tag{2.8}$$

where

$$\Omega_i(t) = \begin{bmatrix} 0 & w_{i3}(t) & -w_{i2}(t) & w_{i1}(t) \\ -w_{i3}(t) & 0 & w_{i1}(t) & w_{i2}(t) \\ w_{i2}(t) & -w_{i1}(t) & 0 & w_{i3}(t) \\ -w_{i1}(t) & -w_{i2}(t) & -w_{i3}(t) & 0 \end{bmatrix} \tag{2.9}$$

and J is the inertia matrix, which is considered to be the same for all spacecraft for simplicity. The collision avoidance constraints are

$$\|\boldsymbol{r}_i(t) - \boldsymbol{r}_j(t)\| \geq R \tag{2.10}$$

for $i, j \in 1 \ldots N$, $i \neq j$, and R is the minimum distance allowed between the centers of spacecraft. The *absolute stay outside* pointing constraints are given by

$$\boldsymbol{z}_k^T \boldsymbol{y}_k \leq \cos\theta_k \tag{2.11}$$

where $k \in 1 \ldots N_c$ identifies a constraint. This condition ensures that the spacecraft vector $\boldsymbol{y}_k$ remains at an angle greater than $\theta_k \in [0, \pi]$ from the inertial vector $\boldsymbol{z}_k$. The vector $\boldsymbol{y}_k$ is the representation in the inertial coordinate frame of the body vector $\boldsymbol{y}_{kB}$. The transformation of coordinates is given by

$$\boldsymbol{y}_k = \boldsymbol{y}_{kB} - 2(\boldsymbol{q}_{i0}^T\boldsymbol{q}_{i0})\boldsymbol{y}_{kB} + 2(\boldsymbol{q}_{i0}^T\boldsymbol{y}_{kB})\boldsymbol{q}_{i0} - 2q_{i4}(\boldsymbol{y}_{kB} \times \boldsymbol{q}_{i0}) \tag{2.12}$$

where $\boldsymbol{q}_{i0}(t)$ and $q_{i4}(t)$ are defined as

$$\boldsymbol{q}_i(t) = [q_{i1}(t), q_{i2}(t), q_{i3}(t), q_{i4}(t)]^T = [\boldsymbol{q}_{i0}(t), q_{i4}(t)]^T$$

The *absolute stay inside* pointing constraints only differ by the sign of the equation

$$\boldsymbol{z}_k^T \boldsymbol{y}_k \geq \cos\theta_k \tag{2.13}$$

The inter-spacecraft *relative stay outside* pointing constraints are given by

$$\hat{\boldsymbol{r}}_{ij}^T \boldsymbol{y}_k \leq \cos\theta_k \tag{2.14}$$

where $\boldsymbol{y}_k$ and θ_k are as above, and $\hat{\boldsymbol{r}}_{ij}(t) = (\boldsymbol{r}_j(t) - \boldsymbol{r}_i(t))/(\|\boldsymbol{r}_j(t) - \boldsymbol{r}_i(t)\|)$ is the unit vector that points from spacecraft i to spacecraft j. Similarly, the *relative stay inside* pointing constraints are

$$\hat{\boldsymbol{r}}_{ij}^T \boldsymbol{y}_k \geq \cos\theta_k \tag{2.15}$$

For this problem, the cost to minimize is

$$J = \sum_{i=1}^{N} \int_0^T |\boldsymbol{T}_i(t)| + |\boldsymbol{M}_i(t)|\, dt \tag{2.16}$$

2.2 The Solution Approach

2.2.1 The Planner

The randomized path planning algorithm consists of Lavalle's randomized dense tree, in particular the bidirectional variant as described in [25]. The algorithm is reproduced here, but interested readers are encouraged to consult references [24] or [25].

RRT-BIDIRECTIONAL($\boldsymbol{x}_i, \boldsymbol{x}_f$)

```
1  T_a.init(x_i); T_b.init(x_f)
2  for j ← 1 to K
3        do x_n ← NEAREST(T_a, α(j))
4           x_s ← CONNECT(x_n, α(j))
5           if x_s ≠ x_n
6              then T_a.add-vertex(x_s)
7                   T_a.add-edge(x_n, x_s)
8                   x'_n ← NEAREST(T_b, x_s)
9                   x'_s ← CONNECT(x'_n, x_s)
10                  if x'_s ≠ x'_n
11                     then T_b.add-vertex(x'_s)
12                          T_b.add-edge(x'_n, x'_s)
13                  if x'_s = x_s
14                     then return Solution
15          SWAP(T_a, T_b)
16 return Failure
```

In this algorithm, T_a and T_b represent trees with a composite trajectory point $\boldsymbol{x}$ at each node (Eq. 2.5). The points $\boldsymbol{x}$ at the nodes are considered at rest, so the only relevant information in these points are positions and attitudes. The two trees T_a and T_b start from the initial and final points of the desired trajectory. At each iteration $\alpha(i)$ generates a random point, and NEAREST($T_a, \alpha(i)$) finds the point in the tree T_a with the minimum distance to this point $\alpha(i)$. The distance is defined as

$$\text{distance}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sum_{i=1}^{N} \|\boldsymbol{r}_{1,i} - \boldsymbol{r}_{2,i}\| + K_a \angle(\boldsymbol{q}_{1,i}, \boldsymbol{q}_{2,i})$$

where $\angle(\boldsymbol{q}_{1,i}, \boldsymbol{q}_{2,i})$ is the angle of an eigen-axis rotation between attitude $\boldsymbol{q}_{1,i}$ and $\boldsymbol{q}_{2,i}$ for spacecraft i, and K_a is a weight that relates translation distance and rotation angle. In the scenarios presented later, $K_a = 6$.

Continuing with the algorithm, CONNECT($\boldsymbol{x}_n, \alpha(i)$) finds the last valid configuration in the "direct motion" from $\boldsymbol{x}_0$ to $\boldsymbol{x}$.

CONNECT($\boldsymbol{x}_0, \boldsymbol{x}_f$)

1 $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_0$

2 **repeat** $\boldsymbol{x}_n \leftarrow$ EXTEND($\boldsymbol{x}_n, \boldsymbol{x}_f$)

3 **until** could not advance $\boldsymbol{x}_n$

4 **return** $\boldsymbol{x}_n$

In our implementation, a *direct motion* is a rest-to-rest straight line translation and eigen-axis rotation of each spacecraft

$$\left.\begin{aligned} \text{trans}(\boldsymbol{r}_{1,i}, \boldsymbol{r}_{2,i}; t) &= \boldsymbol{r}_{1,i} + t(\boldsymbol{r}_{2,i} - \boldsymbol{r}_{1,i}) \\ \text{rot}(\boldsymbol{q}_{1,i}, \boldsymbol{q}_{2,i}; t) &= \boldsymbol{q}_{1,i} \cdot (\boldsymbol{q}_{1,i}^{-1} \cdot \boldsymbol{q}_{2,i})^t \end{aligned}\right\} \quad \begin{array}{c} \forall i \\ t \in [0,1] \end{array} \tag{2.17}$$

where rot is a quaternion interpolation.

If a new node is successfully found by the direct motion, then a branch to this node is added to the tree, and a similar attempt is made to connect the opposite tree to the new node. If the attempt succeeds the algorithm stops and returns a good trajectory, otherwise it continues.

The output of the algorithms consists of a sequence of points from the initial point $\boldsymbol{x}_i$ to the desired target point $\boldsymbol{x}_f$. At these points the spacecraft are at rest, their states are described by position and attitude values, and there is a direct motion to the next point that is guaranteed to satisfy all the constraints.

2.2.2 The Smoother

The resulting trajectory from the planner is then smoothed. However, this trajectory must be described first as full state and input pairs sampled at fixed time-steps. In general these samples will not coincide with the points of the trajectory from the planner, and the spacecraft will not necessarily be at rest at these points in time. In this section the state notation $\boldsymbol{x}(t)$ has been replaced by $\boldsymbol{x}(k)$ which stands for $\boldsymbol{x}(k\Delta T)$, where ΔT is the time-step. The complete trajectory is represented by the sequence of points $\boldsymbol{x}(k), k \in 0 \ldots \lceil T/\Delta T \rceil$. For these points

$$\boldsymbol{x}(k+1) = \boldsymbol{f}(\boldsymbol{x}(k)) \tag{2.18}$$

where $\boldsymbol{f}(\boldsymbol{x}(k))$ is the propagation for time ΔT of the states $\boldsymbol{x}(k)$ for constant inputs $\boldsymbol{T}(k)$ and $\boldsymbol{M}(k)$. To ensure consistency between all elements of $\boldsymbol{x}(k)$ (states and inputs), $\dot{\boldsymbol{r}}(k)$, $\boldsymbol{T}(k)$, $\boldsymbol{w}(k)$ and $\boldsymbol{M}(k)$ must be found that satisfy Eq. 2.18. By using a discrete approximation for the short time interval propagation it is possible to achieve this consistency. The discrete equations for translational dynamics are

$$\begin{bmatrix} \boldsymbol{r}_i(k+1) \\ \dot{\boldsymbol{r}}_i(k+1) \end{bmatrix} = \begin{bmatrix} I & \Delta T I \\ 0 & I \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_i(k) \\ \dot{\boldsymbol{r}}_i(k) \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta T I \end{bmatrix} \boldsymbol{T}_i(k) \tag{2.19}$$

and for the attitude dynamics

$$\boldsymbol{w}_i(k+1) = \boldsymbol{w}_i(k) - \Delta T J^{-1} \boldsymbol{w}_i(k) \times (J \boldsymbol{w}_i(k)) + \Delta T J^{-1} \boldsymbol{M}_i(k) \tag{2.20}$$

and

$$\boldsymbol{q}_i(k+1) = \left[I + \frac{\Delta T}{2} \Omega_i(k) \right] \boldsymbol{q}_i(k) \tag{2.21}$$

where $\boldsymbol{w}_i(k)$ is just the discrete form of Eq. 2.9. Thus to obtain the full $\boldsymbol{x}(k)$ from x and q

$$\dot{\boldsymbol{r}}_i(k) = \frac{\boldsymbol{r}_i(k+1) - \boldsymbol{r}_i(k)}{\Delta T} \tag{2.22}$$

$$\boldsymbol{T}_i(k) = \frac{\dot{\boldsymbol{r}}_i(k+1) - \dot{\boldsymbol{r}}_i(k)}{\Delta T} \tag{2.23}$$

$$\boldsymbol{M}_i(k) = J \frac{\boldsymbol{w}_i(k+1) - \boldsymbol{w}_i(k)}{\Delta T} + \boldsymbol{w}_i(k) \times (J \boldsymbol{w}_i(k)) \tag{2.24}$$

and $\boldsymbol{w}_i(k) = [\tilde{w}_{i1}(k), \tilde{w}_{i2}(k), \tilde{w}_{i3}(k)]^T$, from

$$\tilde{\boldsymbol{w}}_i(k) = 2 Q_i(k)^{-1} \frac{\boldsymbol{q}_i(k+1) - \boldsymbol{q}_i(k)}{\Delta T} \tag{2.25}$$

where

$$Q_i(k) = \begin{bmatrix} q_{i4}(k) & -q_{i3}(k) & q_{i2}(k) & q_{i1}(k) \\ q_{i3}(k) & q_{i4}(k) & -q_{i1}(k) & q_{i2}(k) \\ -q_{i2}(k) & q_{i1}(k) & q_{i4}(k) & q_{i3}(k) \\ -q_{i1}(k) & -q_{i2}(k) & -q_{i3}(k) & q_{i4}(k) \end{bmatrix}$$

The discrete dynamics (Eqs. 2.19–2.21) can then be represented as the equalities

$$\boldsymbol{x}(k+1) - \boldsymbol{f}(\boldsymbol{x}(k)) = 0, k \in 0 \ldots (\lceil T/\Delta T \rceil - 1) \tag{2.26}$$

with pointing and collision avoidance constraints

$$g_n(\boldsymbol{x}(k)) \leq 0, k \in 0 \ldots \lceil T/\Delta T \rceil, n \in 1, \ldots, N_c \tag{2.27}$$

An iteration of the smoothing algorithm consists of finding a perturbation of the trajectory that improves the cost while maintaining the feasibility of the trajectory. The update is achieved by using the first-order Taylor approximation of the constraints and the cost

$$\begin{aligned} &\boldsymbol{x}(k+1) + d\boldsymbol{x}(k+1) - \boldsymbol{f}\left(\boldsymbol{x}(k) + d\boldsymbol{x}(k)\right) \\ &\approx \boldsymbol{x}(k+1) - \boldsymbol{f}(\boldsymbol{x}(k)) + d\boldsymbol{x}(k+1) - \nabla \boldsymbol{f}(\boldsymbol{x}(k))^T d\boldsymbol{x}(k) \\ &= 0, \end{aligned} \tag{2.28}$$

and

$$g_n(\boldsymbol{x}(k) + d\boldsymbol{x}(k)) \approx g_n(\boldsymbol{x}(k)) + \nabla g_n(\boldsymbol{x}(k))^T d\boldsymbol{x}(k) \leq 0 \tag{2.29}$$

for $\|d\boldsymbol{x}(k)\| \leq \epsilon \ll 1$. Note that these are *linear constraints* in the variables $d\boldsymbol{x}(k)$ since $\boldsymbol{f}$, g_n and $\boldsymbol{x}(k)$ are known in advance.

The discrete form of the cost function is

$$J = \Delta T \sum_{k=0}^{\lceil T/\Delta T \rceil} \sum_{i=1}^{N} |\boldsymbol{T}_i(k)| + |\boldsymbol{M}_i(k)| \tag{2.30}$$

which can be rewritten as

$$J = \Delta T \sum_{k=0}^{\lceil T/\Delta T \rceil} \sum_{i=1}^{N} |\boldsymbol{T}_i(k) + d\boldsymbol{T}_i(k)| + |\boldsymbol{M}_i(k) + d\boldsymbol{M}_i(k)| \tag{2.31}$$

which in a linear minimization is equivalent to

$$J = \Delta T \sum_{k=0}^{\lceil T/\Delta T \rceil} \sum_{i=1}^{N} a_i(k) + b_i(k) \tag{2.32}$$

subject to

$$|\boldsymbol{T}_i(k) + d\boldsymbol{T}_i(k)| \leq a_i(k), \ \forall i, k \tag{2.33}$$

$$|\boldsymbol{M}_i(k) + d\boldsymbol{M}_i(k)| \leq b_i(k), \ \forall i, k \tag{2.34}$$

The step to find the perturbation is naturally formulated as a linear program because it is the minimization of a linear function subject to linear equality and inequality constraints (Line 2 in the SMOOTHER-STEP). Here the trajectory is improved as a whole and the process continues in a deterministic fashion. The algorithm is as follows:

SMOOTHER($\boldsymbol{x}$)

1 **for** $j = 1$ **to** M

2 **do** SMOOTHER-STEP($\boldsymbol{x}$)

3 **return** $\boldsymbol{x}$

SMOOTHER-STEP($\boldsymbol{x}$)

1 **for** $i = 1$ **to** N

2 **do** solve linear program:

$$\min \Delta T \sum_{k=0}^{\lceil T/\Delta T \rceil} a_i(k) + b_i(k)$$

subject to

$$\forall k \begin{cases} \begin{bmatrix} I & -\nabla \boldsymbol{f}(\boldsymbol{x}_i(k))^T \end{bmatrix} \begin{bmatrix} d\boldsymbol{x}_i(k+1) \\ d\boldsymbol{x}_i(k) \end{bmatrix} = 0 \\ \nabla g_n(\boldsymbol{x}_i(k))^T d\boldsymbol{x}_i(k) \leq -g_n(\boldsymbol{x}_i(k)) \\ |\boldsymbol{T}_i(k) + d\boldsymbol{T}_i(k)| \leq a)i(k) \\ |\boldsymbol{M}_i(k) + d\boldsymbol{M}_i(k)| \leq b_i(k) \\ |d\boldsymbol{x}_i(k)| \leq \epsilon \end{cases}$$

▷ end of linear program

3 $\boldsymbol{x}_i(k) \leftarrow \boldsymbol{x}_i(k) + d\boldsymbol{x}_i(k), \forall k$

4 $\boldsymbol{q}_i(k) \leftarrow \frac{\boldsymbol{q}_i(k)}{\|\boldsymbol{q}_i(k)\|}, \forall k$

5 $[\dot{\boldsymbol{r}}_i(k),\ \boldsymbol{T}_i(k)] \leftarrow$ inverse of $[\boldsymbol{x}_i(k),\ \boldsymbol{x}_i(k+1)], \forall k,$ from equations (2.22) and (2.23)

6 $[\boldsymbol{w}_i(k),\ \boldsymbol{M}_i(k)] \leftarrow$ inverse of $[\boldsymbol{q}_i(k),\ \boldsymbol{q}_i(k+1)], \forall k,$ from equations (2.25) and (2.24)

▷ end of for

7 **return** $\boldsymbol{x}$

Due to the linear approximations, in general the updated solution in step 3 may violate the constraints by a small amount (less than $O(\epsilon)$, where $0 < \epsilon \ll 1$). Therefore the solution is repaired in lines 4 to 6 to recover the consistency with the constraints of the problem, particularly the unit norm quaternion constraint and the dynamics from Eqs. 2.19–2.21. The inequality constraints are not explicitly repaired. Any violation of these constraints is negligible and does not affect the convergence of the algorithm. To guarantee that the final solution meets the constraints, a small margin is added to the constraints before running the algorithm. For example, the angles and collision radius are increased by a small percentage over the actual values.

In summary, our approach consists of the following steps:

FIND-RECONFIGURATION($\boldsymbol{x}_i, \boldsymbol{x}_f$)

1 s = RRT-BIDIRECTIONAL($\boldsymbol{x}_i, \boldsymbol{x}_f$)

2 **if** $s \neq Failure$

3 **then** discretize s

4 SMOOTHER(s)

5 **return** s

2.3 Examples

This section presents examples of varying complexity. They demonstrate that for simple problems the algorithm generates the expected results, and for harder problems it generates reasonable trajectories. In the figures that illustrate these examples the trajectories are represented by a solid line, each dot represents a time step, and arrows show the direction of movement. The spacecraft are typically shown along the trajectories after every fifth time step. The vectors shown are the X, Y, and Z body axes. The plot axes correspond to the axes of the local inertially fixed frame. Also, the examples that include a sun avoidance (stay-out) constraint show a "red umbrella", with the "handle" representing the vector pointing *toward* the sun, and a cone of rays representing the angle covered by the constraint.

These experiments were run on a Fujitsu T3000 with an Intel Centrino processor at 1.4 GHz and Windows XP, and the algorithms were programmed in C++ and compiled with Microsoft Visual C++ 7.1. The linear solver used was the GLPK library. The computation times for the *planner* ranged from below 1 second for single spacecraft problems like the simple maneuver, to 30-40 minutes for highly constrained problems with 4 spacecraft. The number of iterations of the main loop in RRT-BIDIRECTIONAL ranged from 1 to 10 for simple examples, to between 200 and 500 for the difficult ones. The cut-off in our experiments was 500 iterations, after which the algorithm returned a failure. The computation times for the *smoother* were from below 1 second to 3 seconds, using the interior point solver.

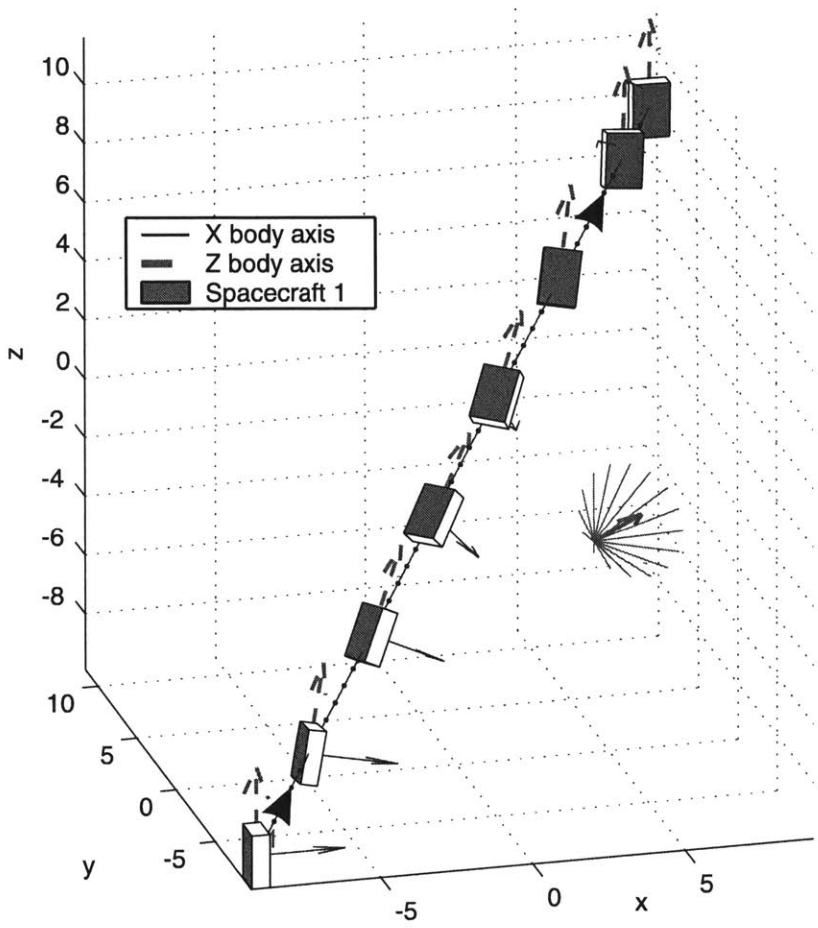


Figure 2.1: Example: Simple maneuver. Simple translation and rotation with sun avoidance constraint.

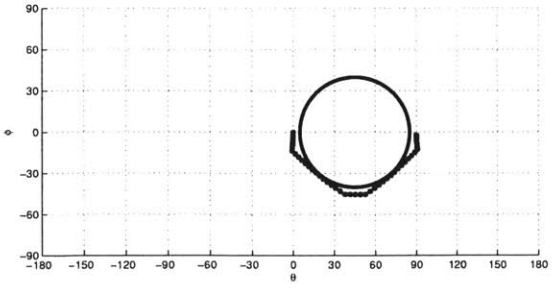


Figure 2.2: Locus of "instrument" vector is tight against sun avoidance constraint

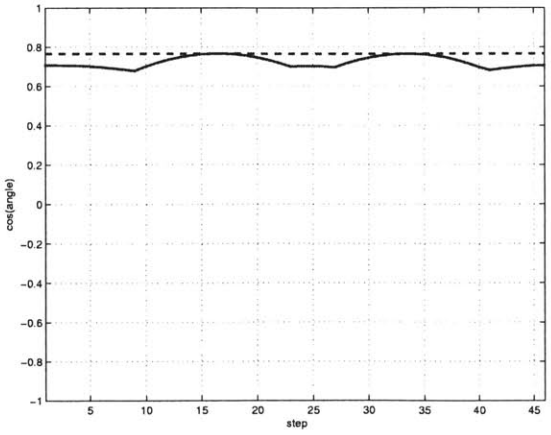


Figure 2.3: Cosine of angle between "instrument" vector and sun vector

2.3.1 Example: Simple Maneuver

Figure 2.1 shows the final trajectory for a simple problem: move a spacecraft from $[-9, -9, -9]^T$ to $[9, 9, 9]^T$ and rotate it 90° about the inertial Z-axis, while avoiding pointing at the sun. The unit vector pointing at the sun is represented in the figure by the vector in the direction of $[1, 1, 0]^T/\sqrt{2}$ surrounded by a 40^o angle cone. The sensitive instrument points in the direction of the body X axis (solid blue in the figure) and it must stay out of the cone. The plot shows that the solver designs a smooth trajectory that skirts this constraint. As expected, the translation path is minimal: a simple straight line. Figure 2.2 shows the locus of the instrument in spherical coordinates as it moves around the sun avoidance constraint. Figure 2.3 shows the same pointing constraint over time, as the cosine of the angle between the instrument and the sun vector, with a dashed line showing the actual constraint. Both figures show that the trajectory is very close to the constraint.

2.3.2 Example: Coupled Maneuver

Figures 2.4 and 2.5 show a slightly more complex example with three spacecraft that demonstrate the interaction between the states of the spacecraft and the coupling constraints. Spacecraft 1 and 2 are initially at positions $[-5, 5, 0]^T$ and $[-5, -5, 0]^T$. Spacecraft 1 must turn 180^o around the Z axis and 90^o around the X axis. Spacecraft 2 only has to turn 180^o around the Z axis. Both must also point their body X axis (solid blue) at the other spacecraft to within 30^o. Spacecraft 3 must end at the same starting position of $[-5, 9, 0]^T$, and point its body X axis at both spacecraft 1 and 2 to within 15^o angle. The vehicles must remain 3.5 units apart to avoid colliding.

The final trajectory in Figure 2.4 shows that spacecraft 2 translates straight from the start to finish while spacecraft 1 moves away just enough to satisfy the collision avoidance constraint. Notice that for spacecraft 1 to point the body X axis at spacecraft 2 while avoiding pointing it at the Sun, it has to move off the X-Y plane. This shows the coupling between relative and absolute pointing constraints, and collision avoidance. Also, when spacecraft 1 moves off the initial alignment, spacecraft

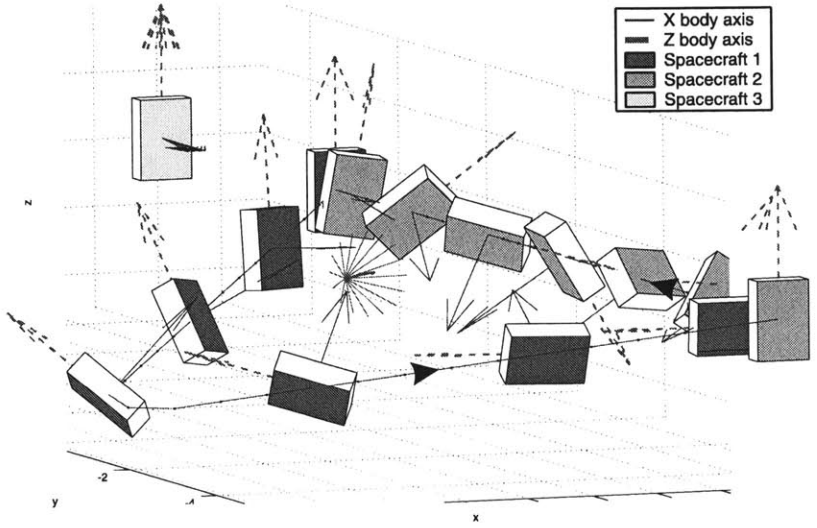


Figure 2.4: Example: coupled maneuver. Shown in detail. Spacecraft 1 and 2 switch places while pointing at each other. Spacecraft 3 points at 1 and 2. They also avoid pointing at the sun.

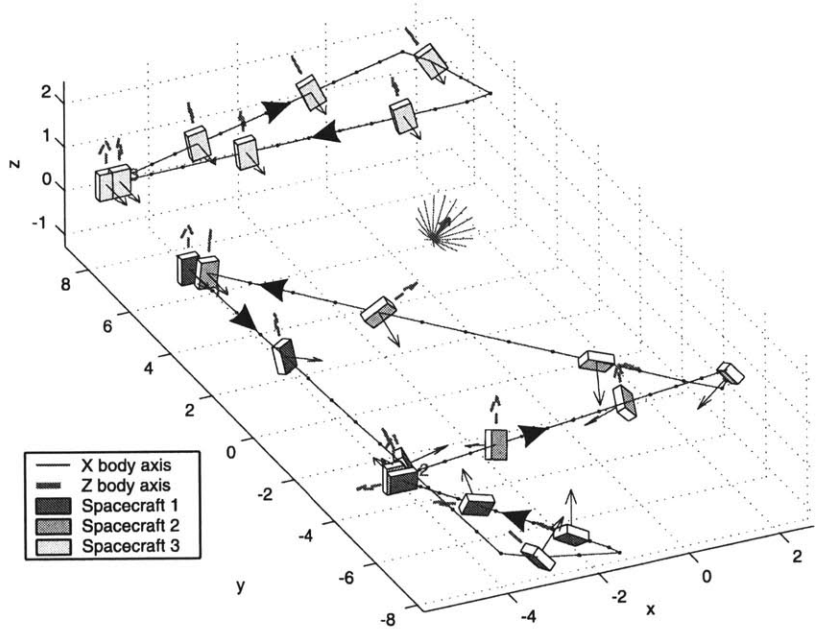


Figure 2.5: Same as Figure 2.4, solution before the smoothing

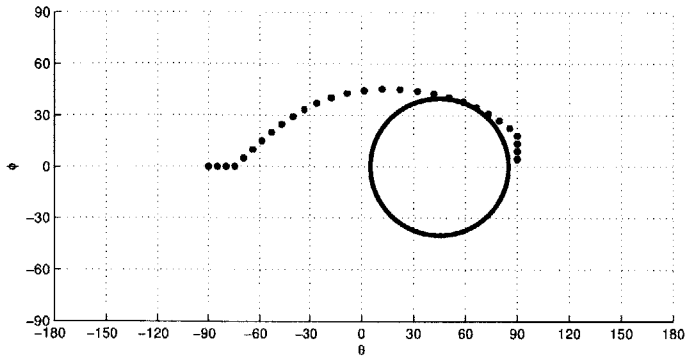


Figure 2.6: Locus of instrument in spacecraft 2 with sun avoidance constraint

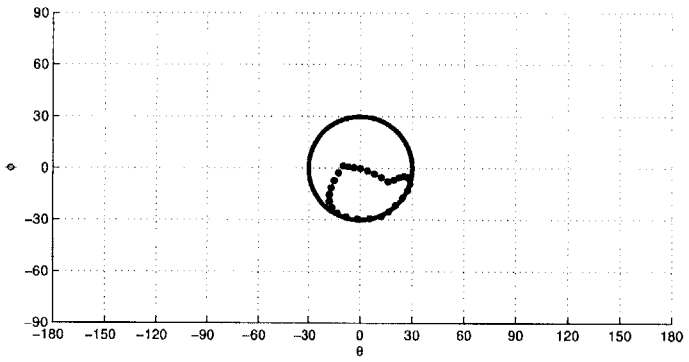


Figure 2.7: Locus of vector pointing from spacecraft 1 to spacecraft 2 (body axis). Must stay inside of cone of ranging device

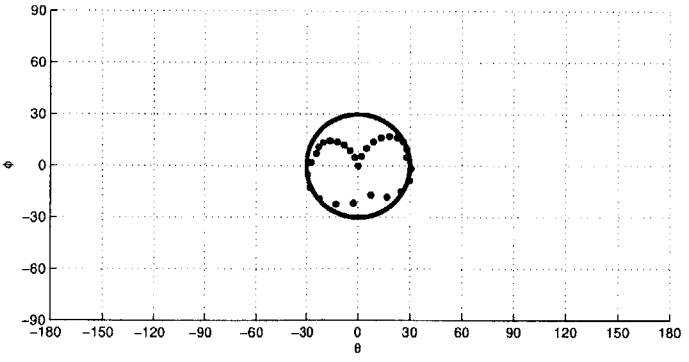


Figure 2.8: Locus of vector pointing from spacecraft 2 to spacecraft 1 (body axis). Must stay inside of cone of ranging device

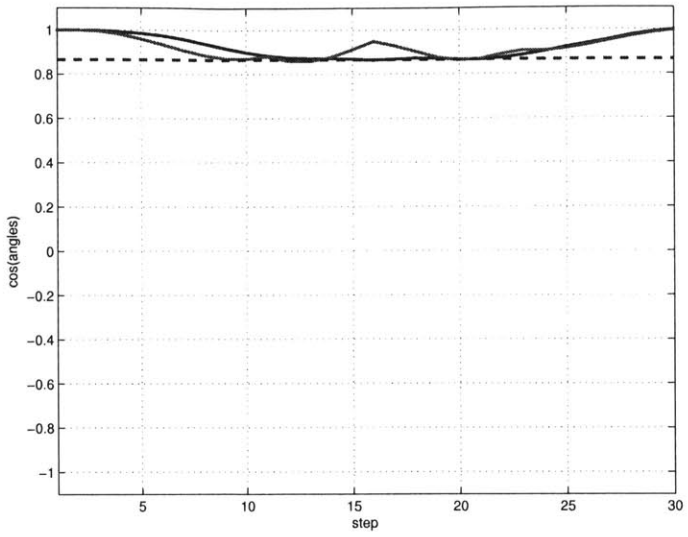


Figure 2.9: Cosines of angles between ranging device vector and relative position of both spacecrafts

3 rotates slightly to keep both spacecraft 1 and 2 inside its specified cone.

The trajectory of Figure 2.4 is based on the trajectory shown in Figure 2.5, which is shown here for comparison. Figure 2.5 shows the trajectory produced by the planner, before the smoothing procedure. As expected, the trajectory is feasible, but clearly suboptimal. All the spacecraft move and rotate away from the target positions, then return to them, which is particularly evident for spacecraft 3. The difference between the figures shows the large changes possible by the smoother given a feasible initial solution.

Figures 2.6 to 2.9 illustrate some of the constraints of this example and how they have a significant impact in the trajectory. Figure 2.6 shows the sun avoidance constraint of spacecraft 2, which is active, and how the spacecraft must maneuver to satisfy it. Figure 2.7 and 2.8 show the relative pointing constraints of spacecrafts 1 and 2 in spherical coordinates. The figures show how restrictive these constraints are, since the instruments must remain pointing within a tight cone through all the trajectory, and it also shows that these constraints are indeed satisfied by the final trajectories. Figure 2.9 shows the same relative pointing constraints as the cosine of the angle between the instrument and the opposite spacecraft. The constraints are very tight, and remain active during a large section of the trajectory.

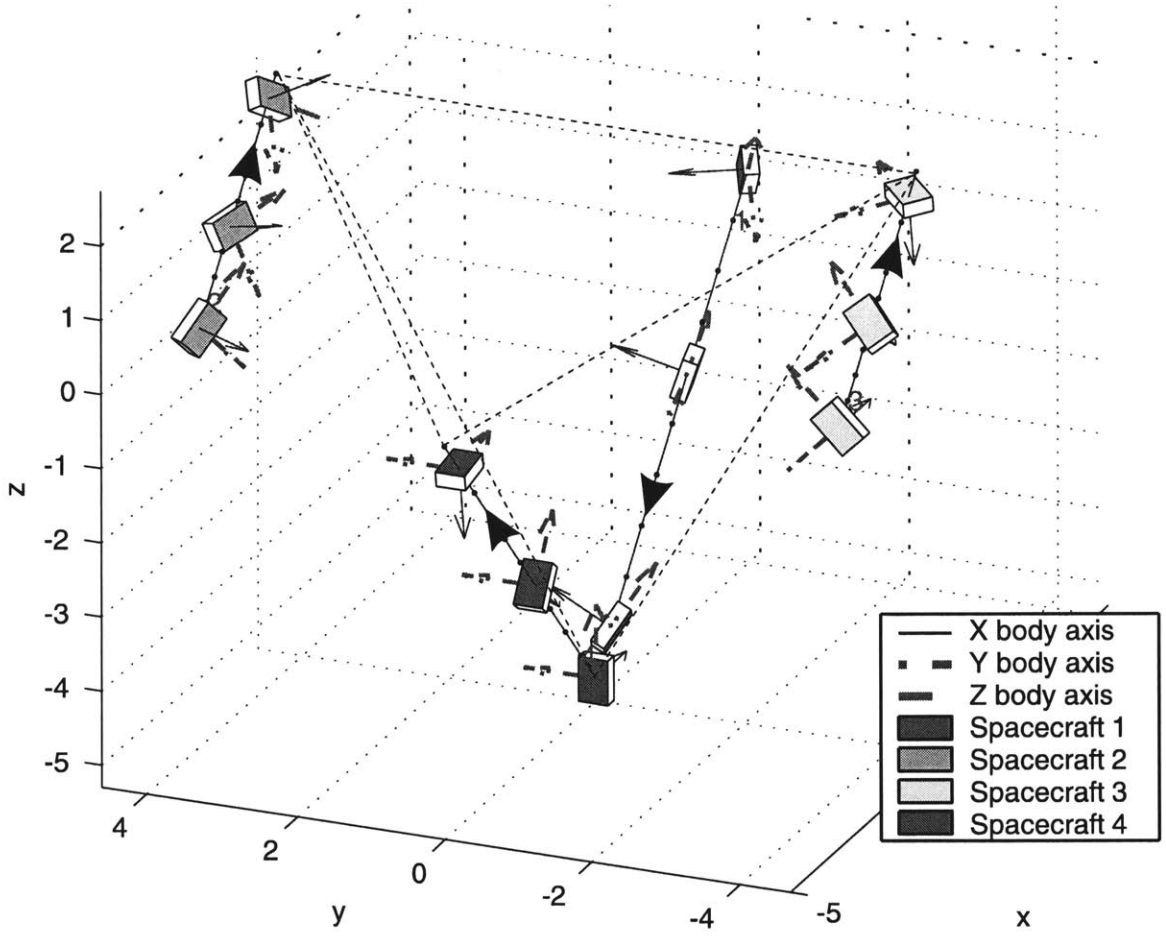


Figure 2.10: Rotate tetrahedral configuration 90^o degrees around Y axis. Pointing constraints remain as before.

2.3.3 Example: Four Vehicles

Figure 2.10 shows the solution to a complex example with four spacecraft. The constraints in this example are as follows: (a) Spacecraft 1, 2 and 3 must point their body Y axis (dashed red) toward spacecraft 4; and (b) they must also point their body X axis (solid blue) to each other in a ring (spacecraft 1 must point at spacecraft 2, 2 to 3, and 3 to 1). These 6 constraints place tight restrictions on the possible movements of the spacecraft which must be closely coordinated. The attitude of spacecraft 4 is not constrained.

The maneuver starts with a tetrahedral formation with spacecraft 1, 2 and 3 in the X-Y plane pointing to spacecraft 4 below. The formation then rotates 90° about the inertial Y axis. The final solution consists of simple straight line translations for all 4 spacecraft, with minimal rotations toward the desired final attitudes, consistent with an optimal maneuver. The constraints are always met.

2.4 Comparison with a Nonlinear Optimal Control Technique

Section 2.2 presents one algorithm for solving the spacecraft reconfiguration problem, but since this problem can be posed as a nonlinear optimal control problem, there are various ways of solving it. One tool that can be used to solve this problem is the software package DIDO introduced in Chapter 1. However, since this is a very hard problem to solve, it is reasonable to expect that a general purpose nonlinear optimal control tool like DIDO will take too long to solve the problem. This section shows that although the spacecraft reconfiguration problem can be formulated and solved by DIDO, it can only do so for small or simple instances of the problem.

2.4.1 Problem Formulation with DIDO

The spacecraft reconfiguration problem formulation with DIDO is not very different from the formulation of section 2.1. The problem can be posed directly from the

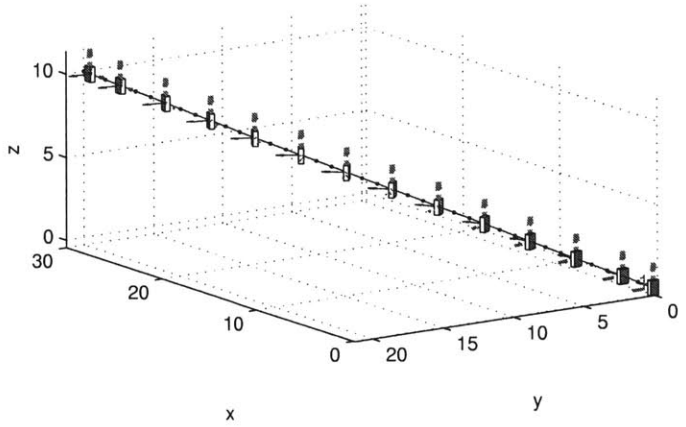
nonlinear optimal control formulation, since DIDO handles the details. The state variables are the same as in Eqs. 2.1 to 2.5, while the dynamics and constraints remain the same as Eqs. 2.6 to 2.15. The only difference is the cost, which has been changed to minimizing

$$J = \sum_{i=1}^{N} \int_0^T \|\boldsymbol{T}_i(t)\|^2 + \|\boldsymbol{M}_i(t)\|^2 \, dt \tag{2.35}$$

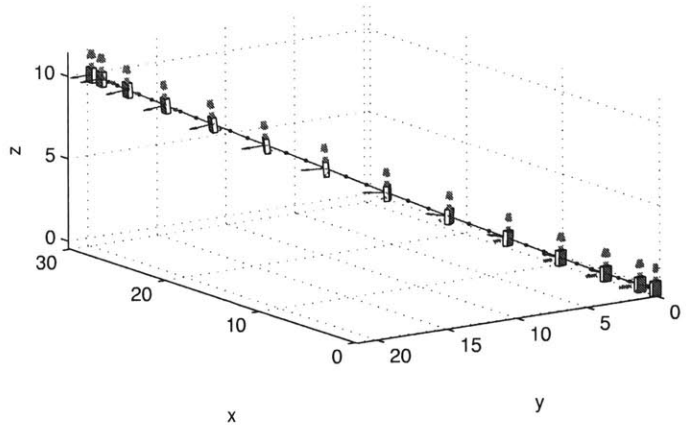
because the absolute value is a non-smooth function that presented an additional difficulty for DIDO. The smoother results shown in this section for comparison have been changed to used this cost function as well. The initial solutions were generated using the planner, and besides the trajectories they also provided the maximum time T for all the examples.

2.4.2 Reconfiguration Problem for Different Initial Solutions and Problem Sizes

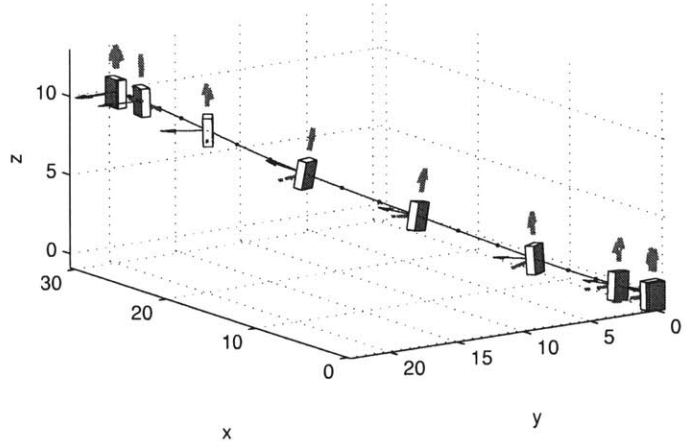
Example A is a simple translation from position $[0, 0, 0]^T$ to $[30, 20, 10]^T$ with a rotation of 90 degrees around the Z body axis. Example B is a similar translation with 90 degrees rotation, only from $[0, 0, 0]^T$ to $[15, 10, 5]^T$. Figure 2.11(a) shows a near optimal trajectory generated for Example A by the planner (Plan). Figure 2.11(b) is the trajectory produced by the *smoother* based on the plan. This trajectory has a cost of 0.430825 and was smoothed after 18 seconds. The running times of the smoother for small cases like this (1 spacecraft) are usually lower, but in this case the tolerance of the algorithm was increased to produce a better cost. Comparatively, Figure 2.11(c) shows the trajectory produced by DIDO, with 28 nodes and no initial guess after 140 seconds, and with a cost of 0.3681. The lowest cost from all DIDO runs for this example was of 0.2845. If this is used as a the best reference, the smoother trajectory has a cost 51% worse than the best, and the DIDO trajectory has a cost 30% worse. Figure 2.12(a) shows the trajectory generated by the planner for Example B. The figure shows a trajectory that is clearly suboptimal, with a change



(a) Plan before smoothing

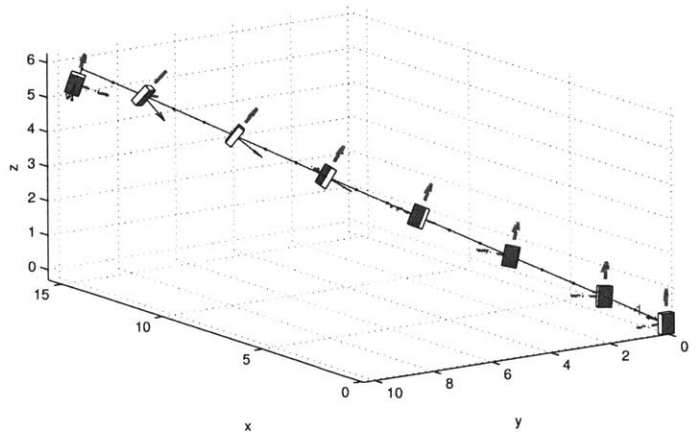


(b) Trajectory produced by the *smoother*

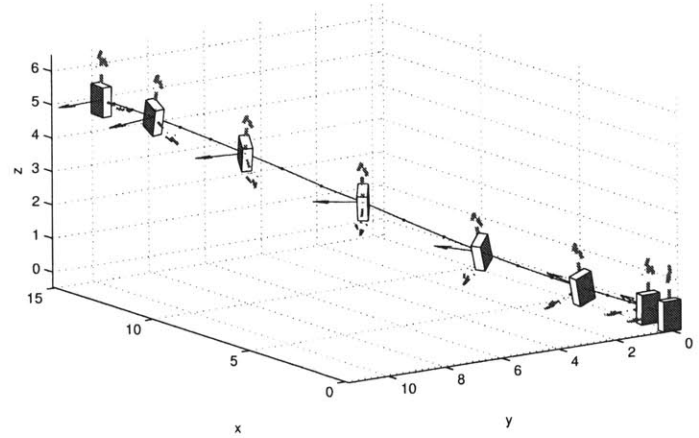


(c) Trajectory produced by DIDO

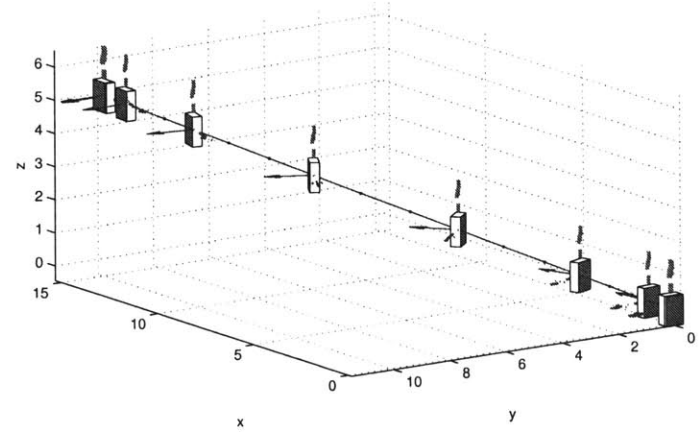
Figure 2.11: Plan and smoothed trajectories for Example A



(a) Plan before smoothing



(b) Trajectory produced by the *smoother*



(c) Trajectory produced by DIDO

Figure 2.12: Plan and smoothed trajectories for Example B

of direction that can be seen in the top left corner. The slew maneuver is also visibly suboptimal. However, the corresponding trajectory produced by the *smoother*, shown in Figure 2.12(b), although not optimal, has smoothed the event in the top left corner, and also the slew maneuver. This trajectory has a cost of 0.50367 which is 26% percent worse than the best cost of 0.3971 obtained by DIDO for this example, and was found after 14 seconds. Figure 2.12(c) shows the trajectory produced by DIDO also with 28 nodes and no initial guess, after 230 seconds, and with a cost of 0.3987, less than 1% worse than the best.

This results give a short glimpse of the characteristics of the trajectories found by the *planner/smoother* combination versus those found by DIDO. DIDO has consistently a better cost, but is much slower than the smoother (the planner amounts to less than a second for this configurations). Furthermore, the results shown here for the smoother are aimed to produce a solution with good cost, but this algorithm can be adjusted to produce a solution much faster, perhaps with larger formations where the computation time would be too large otherwise. In contrast, DIDO cannot be adjusted so it is bound to produce good solutions with poor computation time. However, the real difference between DIDO and the *planner/smoother* combination is in the reliability.

Figure 2.13 and Figure 2.14 show the execution times for the optimizations of Examples A and B with DIDO for different number of nodes. In these figures the runs that failed, either because they took too long (cut-off at 500 seconds) or just because the optimization did not converge, are represented with a value of 510, just so all the results can be seen in the same graph. Figure 2.13 shows that the running time is proportional to the number of nodes, and runs with a large number of nodes (over 30) are more likely to fail. This figures does not show a clear advantage of running DIDO with an initial guess versus without it. Figure 2.14 looks different, even though the basic problem is not, but mostly because here the initial guess is not as good as with Example A (compare figures 2.11(a) and 2.12(a)). As a result, DIDO finds a solution with an initial guess only for a low number of nodes, and even in this case the computation time is worse. In general the computation times, although

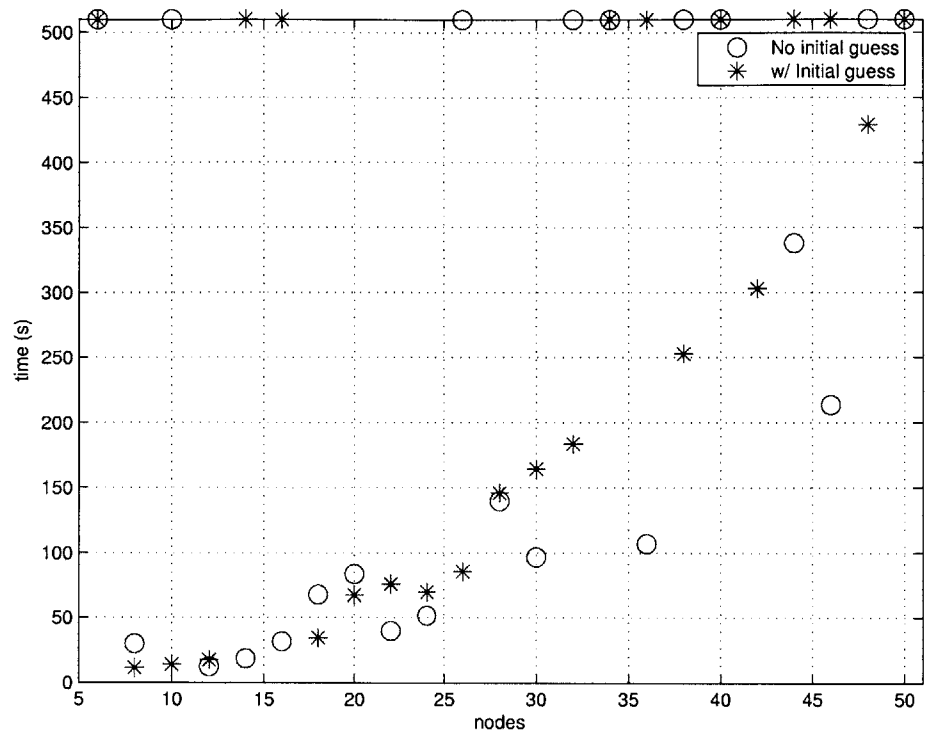


Figure 2.13: Performance of DIDO for Example A. Failed runs are represented with over 500 seconds

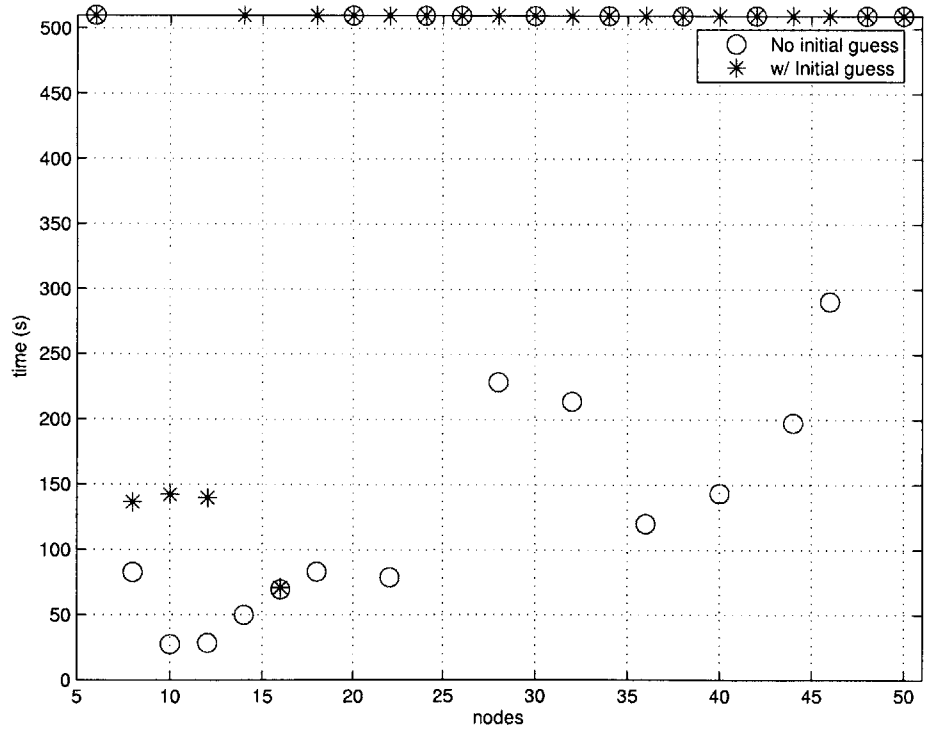


Figure 2.14: Performance of DIDO for Example B. Failed runs are represented with over 500 seconds

somewhat proportional to the number of nodes, are erratic. These two figures suggest that with DIDO, no initial guess is better than a mediocre one.

Figure 2.15 shows the results for Example C. This example, similar to Examples A and B, consists of a translation from $[0, 0, 0]^T$ to $[30, 20, 10]^T$ with a rotation of 90 degrees around the Z axis. This example has an additional pointing constraint, which consists of limiting the Z body axis to point at the inertial Z axis to within 45 degrees. Example C is trivial, since the optimal solution satisfies the constraint which is not active, but it shows the behavior of DIDO with additional constraints, even if trivial ones. With the addition of the constraint the execution times degrade compared with Example A and the solver fails to find a solution for most cases above 30 nodes. The runs without an initial solution are also visibly worse than those of Example B. The smoother stops with a solution after 4 seconds.

Figure 2.16 shows the results for Example D. This example is similar to Example C before, but the constraint is slightly different. In this case the spacecraft is limited to point the X body axis outside of the inertial direction denoted by the vector $[\sqrt{2}, \sqrt{2}, 0]^T/2$. This constraint is in the middle of the path that would be otherwise be optimal and therefore should be active in the final solution. As a result, this constraint makes the problem more difficult than the previous examples. The computational results in Figure 2.16 show that the performance of DIDO degrades even further from before. In particular, the performance without initial guess is much worse. One possible explanation of this is that the initial guess developed internally by DIDO may not take into account the constraint and thus the algorithm has more difficulty converging. In this case the runs with an initial guess fare much better than those without it. The smoother stops with a solution after 20 seconds.

Figure 2.17 shows the results for Example E, which is a combination of the previous two Examples C and D. It consists of a simple translation and rotation, but with both the stay-inside and the stay-outside pointing constraints. This example is more difficult, not only because it increases the number of constraints, but also because the combination of the constraints restricts the space of feasible solutions. This means that to satisfy the stay-outside pointing constraint the spacecraft must rotate off the

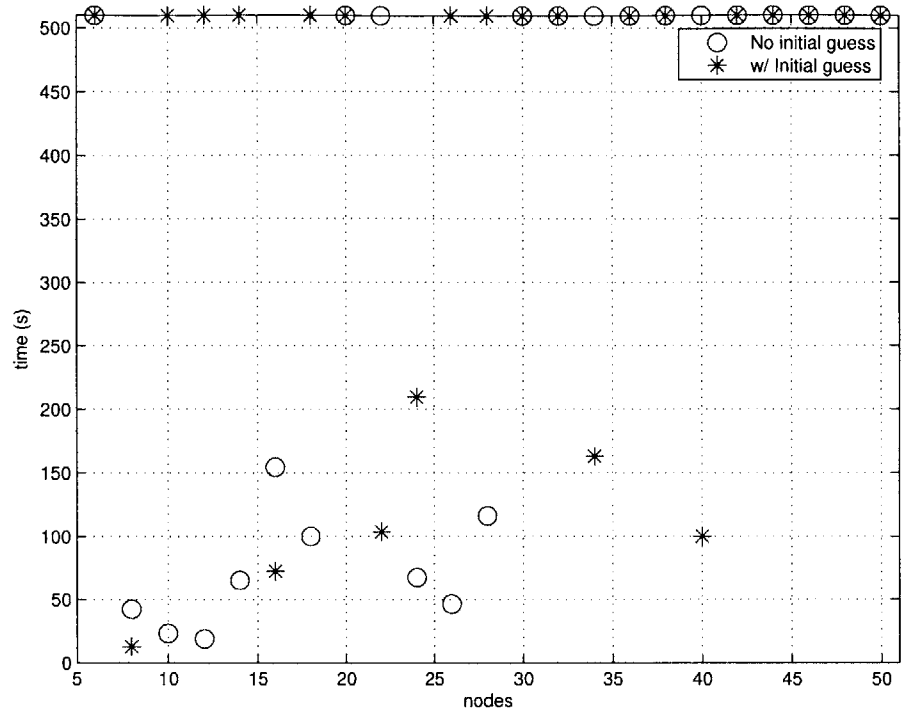


Figure 2.15: Performance of DIDO for Example C (with stay-inside constraint). Failed runs are represented with over 500 seconds

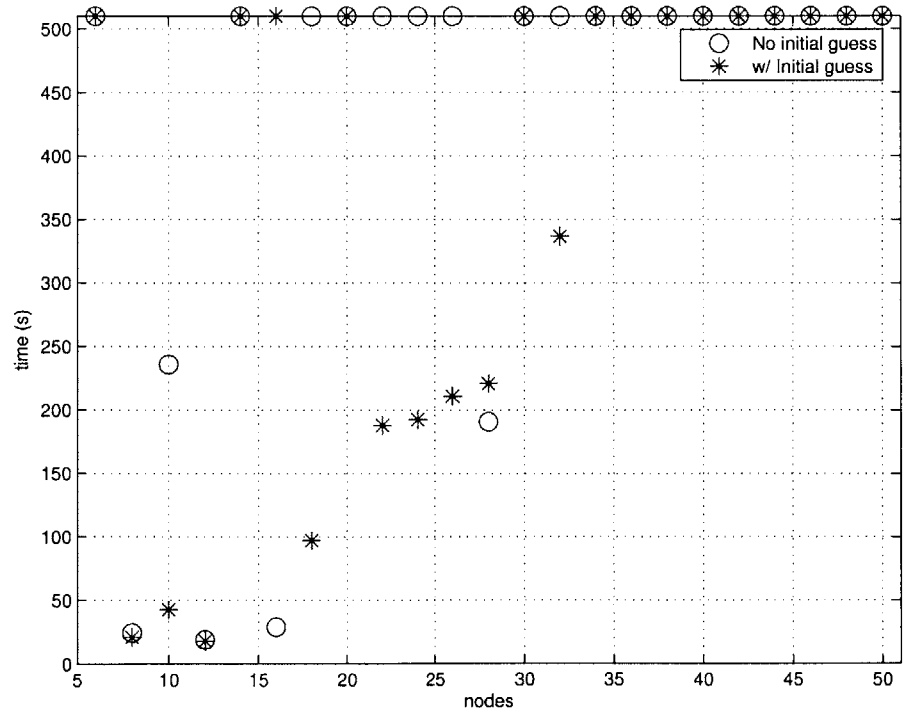


Figure 2.16: Performance of DIDO for Example D (with stay-outside constraint). Failed runs are represented with over 500 seconds

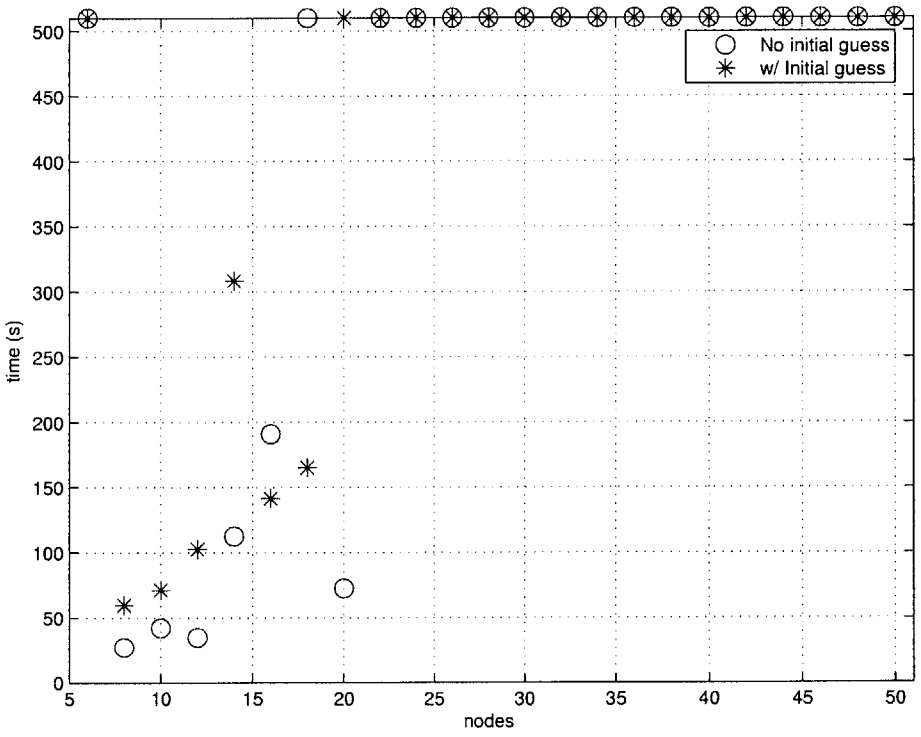


Figure 2.17: Performance of DIDO for Example E (with both stay-inside and stay-outside constraints). Failed runs are represented with over 500 seconds

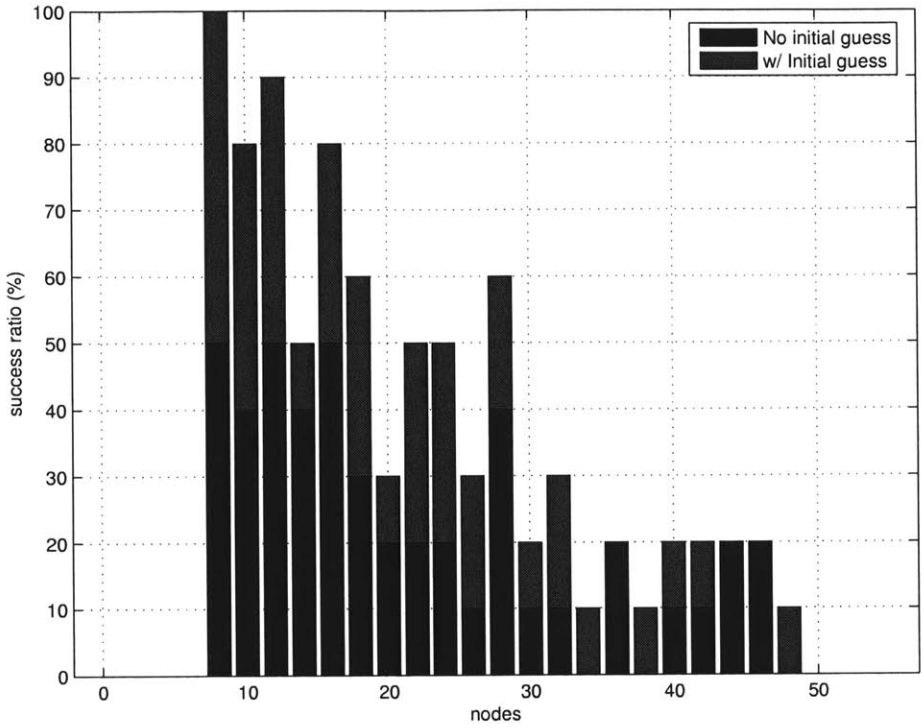


Figure 2.18: Success ratio for DIDO for the examples shown here

$X-Y$ plane but still manage to keep the Z body axis within an angle of the inertial Z to satisfy the stay-inside constraint. Figure 2.17 shows that DIDO fails in all cases with more than 20 nodes. Remarkably, with a low number of nodes DIDO still manages to find solutions both with and without an initial guess. The final smoothed solution was found after 19 seconds.

In general, Figures 2.13 to 2.17 show that:

- The DIDO solution times are erratic. In most cases there is no visible trend in the performance.
- That said, higher number of nodes usually mean worse time and more failures.
- With a bad or non-smooth initial guess like in Example B, DIDO performs worse in time and number of failures than with no initial guess.

Figure 2.18 shows the percentage of successful solutions per number of nodes, with and without an initial guess. This plot shows clearly that the success ratio

degrades with the number of nodes. DIDO found solutions to all the examples with 8 nodes, and the range from 8 to 12 nodes gave good results, with 80% or more of solutions found. It can also be seen in the plot that there is no clear advantage to having an initial guess *on average*, although the previous series of plots showed that in constrained cases the initial guess is an advantage, and that poor initial guesses are a clear disadvantage. In terms of reliability, DIDO is worse than the *planner/smoother* combination since the *smoother* is feasible by design therefore it will always have a valid solution, provided that the *planner* finds one.

2.5 Attitude Maneuver with 1 Spacecraft

This example is a reproduction of the attitude control problem introduced by Frazzoli *et al.* [10]. This problem consists of finding the attitude maneuver between two configurations with one spacecraft, without regard to translation. The article proposes a solution based on Rapidly-exploring Random Trees, also based in LaValle's algorithm, and thus is similar to the *planner* algorithm in section 2.2.1.

The problem consists of rotating a spacecraft with a telescope to point at a new target. In this particular case, the objective is to rotate the spacecraft 90 degrees counter-clockwise about the X inertial axis. The spacecraft is constrained not to point the telescope (Z body axis) within 30 degrees from any bright object (Sun, Earth and Moon) or the star tracker (Y axis) within 20 degrees of the Sun. The antenna (X body axis) must always point within 60 degrees from Earth. The inertial vectors are $[\sqrt{2}, -\sqrt{2}, 0]^T/2$ toward Earth, $[\sin 30, \cos 30, 0]^T$ toward the Moon, and $[0, 1, 0]^T$ toward the Sun. In the original problem the radiator (-Z body axis) cannot point toward the Sun *for more than 120 seconds.* This constraint is an integral constraint that our algorithm cannot include. Instead, for this experiment the radiator is constrained to stay at least 20 degrees away from the line-of-sight to the sun.

The solution is generated by the randomized planner in less than one second, which similar to the computation reported by Frazzoli. The final trajectory shown in Figure 2.19 and Figure 2.20 is the result produced by the *smoother.* These figures show

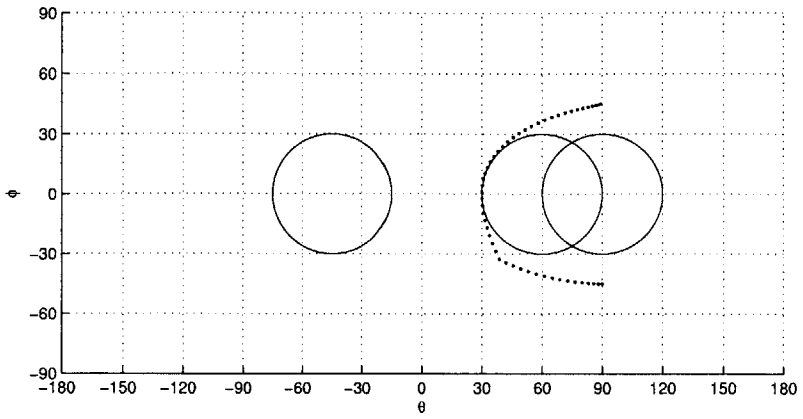


Figure 2.19: Locus of Telescope in Frazzoli's example must stay outside of bright objects

the locus of the telescope and the antenna as the spacecraft follows the trajectory. Figure 2.19 shows that the stay-outside pointing constraint is active, and Figure 2.20 shows that the stay-inside pointing constraint is also satisfied.

DIDO fails to find a good solution when provided with an initial solution, and without an initial solution it finds a solution but it is infeasible. The difficulty with DIDO is that the constraints are enforced at the nodes and if the separation between two nodes is large enough then the trajectory goes over the constraint.

This experiment shows that the *planner* is general enough to solve not only 6 DOF problems with multiple spacecraft, but also to solve attitude maneuvers for 1 spacecraft, with computation times similar to those of algorithm that is specific to this problem [10].

2.6 Summary

Designing spacecraft reconfiguration maneuvers is challenging because it includes non-linear attitude dynamics, difficult non-convex constraints, and high dimensionality ($6N$ DOF) due to coupling of the multiple spacecraft states in the constraints. This chapter presented a method that can solve for reconfigurations for up to 4 spacecraft. The essential feature of this method is the separation into a simplified path planning

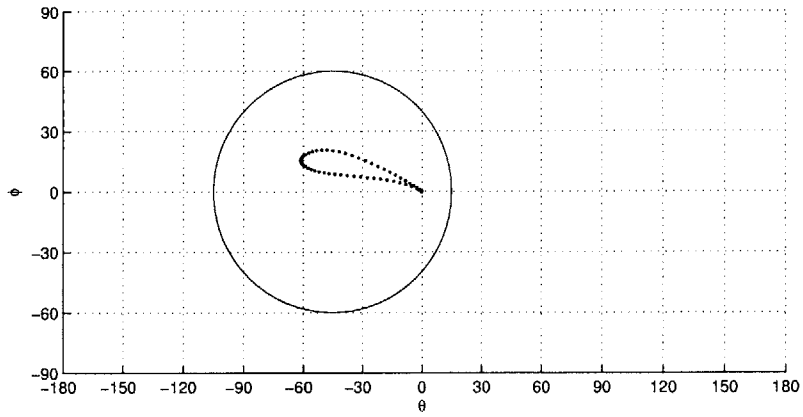


Figure 2.20: Locus of Antenna in Frazzoli's example must point toward Earth

problem without differential constraints to obtain a feasible solution, which is then improved by a smoothing operation. The first step is solved using Rapidly-exploring Random Trees [24]. The smoother consists of an optimization by iteratively solving a linear program using a linearization of the cost function, dynamics, and constraints about the initial feasible solution. The examples demonstrated the validity of the approach and also showed that the algorithm can solve problems with four spacecraft with several complex pointing restrictions.

The approach presented in this chapter is not the only way to solve the spacecraft reconfiguration problem. Since it can be formulated as a nonlinear optimal control problem, it is possible to use a nonlinear control tool to solve it. This chapter uses the software DIDO for this purpose and compares its performance with the *planner/smoother* method. The results show that DIDO did not seem to take advantage of a feasible initial guess but on some constrained problems, and in cases where the initial guess was not smooth it performs better without it. For problems with more than one spacecraft DIDO failed to find any solutions. And even for simple problems with one spacecraft and no more than two constraints, although the solutions DIDO found had better cost, this software was much slower and less reliable than the *planner/smoother*.

Chapter 3

The New Connection Functions

3.1 Introduction

The experiments in the previous chapter showed that the planner found solutions for problems with up to four spacecraft in approximately forty minutes. These are good results compared with previous solutions to similar problems, but better performance is required for real-time applications. Practical solution times of about one minute for configurations from 5 to 10 spacecraft would enable the technique to be used for missions such as the Terrestrial Planet Finder [28]. The planner is the key stage to target to reduce the solution time since, compared to it, the smoother finds solutions in no more than ten seconds for the examples shown in Chapter 2.

Randomized algorithms, of which Rapidly-exploring Random Trees are a variant, were described in Chapter 2 as a means of solving path planning problems. Randomized algorithms perform well in general, but they have been shown to perform poorly for problems that contain difficult regions such as "narrow passages" (i.e., the free space has sections that are long and narrow). For example, in the spacecraft reconfiguration problem of Chapter 2 these type of regions occur whenever the problem has absolute and relative stay-inside pointing constraints. To see why stay-inside pointing constraints create "narrow passages", imagine two spacecraft that must do a complex maneuver, while one of them must point to the other within a narrow angle. The maneuver of spacecraft may involve complex long translations and slew maneuvers,

but the attitude of the second spacecraft must be tightly coordinated with the state of the first spacecraft, and has a very limited range of valid motions. The second spacecraft (or both) can be viewed as moving through a corridor that is long (the whole maneuver) but also narrow (the valid attitude of the 2nd spacecraft). Because the randomized algorithms only generate few sample point in those areas, finding a path through these regions can be very difficult [19]. Solving hard problems that have these difficult regions has generated a large amount of interest, and as a result, numerous methods have been developed to aid the randomized algorithms developed before the RRT's. Two of the best known of these methods are the bridge test and the Gaussian sampling strategies [17, 4]. The common idea behind these methods is that they exert more effort in generating the random sample points. In the basic case, every valid random sample is added to the search tree, while with the new methods select the points with more carefully. In particular, by collecting information about the environment, more points are selected in the difficult regions of the free space. As a result the search trees are typically smaller, but grow more efficiently. However, these new methods cannot be applied directly to Rapidly-exploring Random Trees, which rely on the random samples being evenly distributed over the search space to achieve good performance. This ensures that the search tree is extended toward the unexplored space [24]. Nonetheless, since the sampling methods have been successful in increasing the performance of other randomized algorithms, it is likely that similar methods can be found that are tailored to the characteristics of Rapidly-Exploring Random trees. There has already been some research in this direction, for instance, look at special sampling strategies to improve the expansion of the RRT's [30]. This chapter presents such an extension to the basic RRT algorithm, although in a different direction.

Since the strategy of sampling evenly used by the Rapidly-exploring Random Tree is key to its performance, the approach taken in this chapter is to leave that aspect intact, and instead focus a different element of this algorithm. The element investigated here is the *connection* function, called CONNECT in section 2.2, which in general attempts to connect a point of the search tree with a newly generated sample

point. The connection function in the basic RRT is simple and fast, and only consists of extending a direct path between two points and stopping if an obstacle is found. This simplicity is well suited to the original approach of RRT which is to concentrate on distributing samples and exploring the whole space as fast as possible, for which a complex connection function is not worth the time penalty required to compute it. This thesis proposes that by adding some complexity to the connection function it is possible to improve the performance of the whole algorithm. The new connection function, by collecting additional local information about the free space, is better at connecting two points in difficult cases with obstacles and narrow passages. This combination of the strength of the basic RRT in searching the space on the global scale, and a connection function that searches more on the local scale, could improve the performance of this path planning algorithm.

This chapter proposes two new algorithms that use this principle to improve the basic connection function which, in turn, is shown to significantly improve the RRT algorithm. One algorithm is based on probing the local free space with a number of random trials, and the other is based on artificial potential functions. When the new connection functions encounter an obstacle, instead of just stopping, they use information about the nearby obstacles to try to continue in a valid direction. Both of these algorithms are more flexible than the simple connection function because they take into account the obstacles, and they are faster than a full path planner.

3.2 The New Connection Functions

The basic form of the connection function, introduced in section 2.2 as CONNECT, remains the same

CONNECT($\boldsymbol{x}_0, \boldsymbol{x}_f$)
1 $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_0$
2 **repeat** $\boldsymbol{x}_n \leftarrow$ EXTEND($\boldsymbol{x}_n, \boldsymbol{x}_f$)
3 **until** could not advance $\boldsymbol{x}_n$
4 **return** $\boldsymbol{x}_n$

The EXTEND function is changed to POTENTIAL-EXTEND or RANDOM-EXTEND to use the potential connection function or the random connection function respectively.

3.2.1 The Potential Connection Function

The artificial potential function consists of a continuous function that decreases monotonically in the direction of the goal, (e.g., a *distance* function) [20]. Following the negative gradient of the function generates a path to the goal. Obstacles are then introduced as high *peaks* in the potential function, so that they effectively repel any search that follows the gradient. In ideal circumstances, it is possible to find a path descending to the goal while avoiding obstacles from every point in space. Artificial potential functions were originally developed to solve path planning problems, and gained much appeal due to their simplicity and computational efficiency [20, 43, 12]. In practice however the gradient descent method does not always work because the sum of the distance function plus the high peaks create local minima in the potential field [23]. Additionally, since the obstacles are not hard constraints, but just high peaks in the potential function, they can be violated by the method in some circumstances. Another difficulty with this method is that some path planning problems do not have an obvious formulation with potential functions, or have functions with too many singular points or local minima. For instance, robot path planning problems that involve many obstacles described as complex polygonal shapes are in this category. However, for a path planning problem without differential constraints and with a moderate number of obstacles that are easy to formulate as potential sources (e.g., constraints such as $g(\boldsymbol{x}) \leq 0$), the new method proposed in this section significantly improves the performance of the Rapidly-exploring Random Trees.

The method consists of replacing the CONNECT function in the path planner by a search with a potential function based on a distance metric

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) \tag{3.1}$$

with the obstacles and other constraints represented by inequality constraints of the form

$$g_{min,i} \leq g_i(\boldsymbol{x}) \leq g_{max,i} \tag{3.2}$$

for $i \in 1, \ldots, N$ the number of constraints, $g_{min,i}$ and $g_{max,i}$ are constants, and $g_{min,i} = g_{max,i}$ to represent equality constraints. The search attempts to find a sequence of feasible points that decreases the distance to the target point. This search is posed as a nonlinear optimization problem, and is solved with a feasible sequential optimization method. By posing a step of the search as a nonlinear optimization problem, there is no need to create a single potential function with peaks for constraints. The distance metric and constraints functions a defined separately, and the nonlinear optimizer hides the mechanism by which they are combined. Since the optimization method is feasible, it ensures that the constraints are properly handled and never violated, therefore the intermediate solutions form a valid trajectory. Each new intermediate solution should also be restricted to be within a certain distance of the previous one, to ensure that the trajectory between them is also valid. The potential extend function consists of

POTENTIAL-EXTEND($\boldsymbol{x}, \boldsymbol{x}_f$)

1 Solve nonlinear program:

$\min_{d\boldsymbol{x}} d(\boldsymbol{x} + d\boldsymbol{x}, \boldsymbol{x}_f)$

subject to

$g_{min,i} \leq g_i(\boldsymbol{x} + d\boldsymbol{x}) \leq g_{max,i}, \forall i$

$\|d\boldsymbol{x}\| \leq \epsilon$

▷ End of nonlinear program

2 $\boldsymbol{x}_{best} \leftarrow \boldsymbol{x} + d\boldsymbol{x}$

3 **return** $\boldsymbol{x}_{best}$

In this function $\boldsymbol{x}_i$ and $\boldsymbol{x}_f$ are initial and goal points, $\boldsymbol{x}$ is the temporary point that moves toward the goal by $d\boldsymbol{x}$ increments. The norm of the $d\boldsymbol{x}$ increments is limited to be less than ϵ to ensure the feasibility of the trajectory between two adjacent points.

The numerical experiments in this chapter were done with the custom sequential linear programming method presented in Chapter 2. This method is based on solving

a sequence of linear programs with linearized constraints. Other feasible sequential optimization methods such as FPSQP or CFSQP could also be used here [27, 53].

3.2.2 The Random Connection Function

This search samples the space in the neighborhood around the end point of the trajectory and chooses the valid point with lowest distance to the goal as the next point in the connection. There are two key parameters in this randomization: the radius R of the neighborhood and the number N_{trials} of good trials. A good trial means a trial that is a valid connection and decreases the distance to the goal. The radius of the neighborhood determines the extension of the search, and the number of trials determines the amount of knowledge collected. It can be expected that increasing both the radius and the number of trials would increase the effectiveness of the connection at the expense of computations. A large increase in the neighborhood radius makes the connection function a randomized path planner in itself. Increasing the number of trials tends to duplicate the potential connection function, because this number is proportional to the amount of "information" of what is the best direction to go. Since the potential function has perfect knowledge about the local neighborhood, it is reasonable to expect that as the random function gathers more information, its behavior will approach asymptotically that of the potential function. This result will be illustrated in section 3.3. The random extend function consists of

RANDOM-EXTEND($\boldsymbol{x}_i, \boldsymbol{x}_f$)

1 $\boldsymbol{x}_{best} \leftarrow \boldsymbol{x}_i$
2 **for** $j \leftarrow 1$ **to** N_{trials}
3 **do repeat** $\boldsymbol{x}_n \leftarrow$ NEW-NEIGHBOR($\boldsymbol{x}_i, R$)
4 $d_n \leftarrow d(\boldsymbol{x}_n, \boldsymbol{x}_f)$
5 **until** $d_n \leq d(\boldsymbol{x}_i, \boldsymbol{x}_f)$
6 **if** VALID($\boldsymbol{x}_n$) and $d_n < d(\boldsymbol{x}_{best}, \boldsymbol{x}_f)$
7 **then** $\boldsymbol{x}_{best} \leftarrow x_n$
8 **return** $\boldsymbol{x}_{best}$

In this algorithm NEW-NEIGHBOR generates a random state in the neighborhood within a radius R of x_n, and VALID is true only if $\boldsymbol{x}_n$ does not violate any constraint.

3.3 Illustration of Connection Functions with Simple 2D Path Planning Example

The direct, potential function, and random connection functions are illustrated in this section with a simple path planning problem. The computation times of these algorithms is compared as well. The path planning problem consists of a 2D path planning problem with ellipsoidal obstacles. The advantage of representing simple obstacles as ellipsoids is that they can be easily described by the inequalities required by the potential connection function. The distance metric consists of

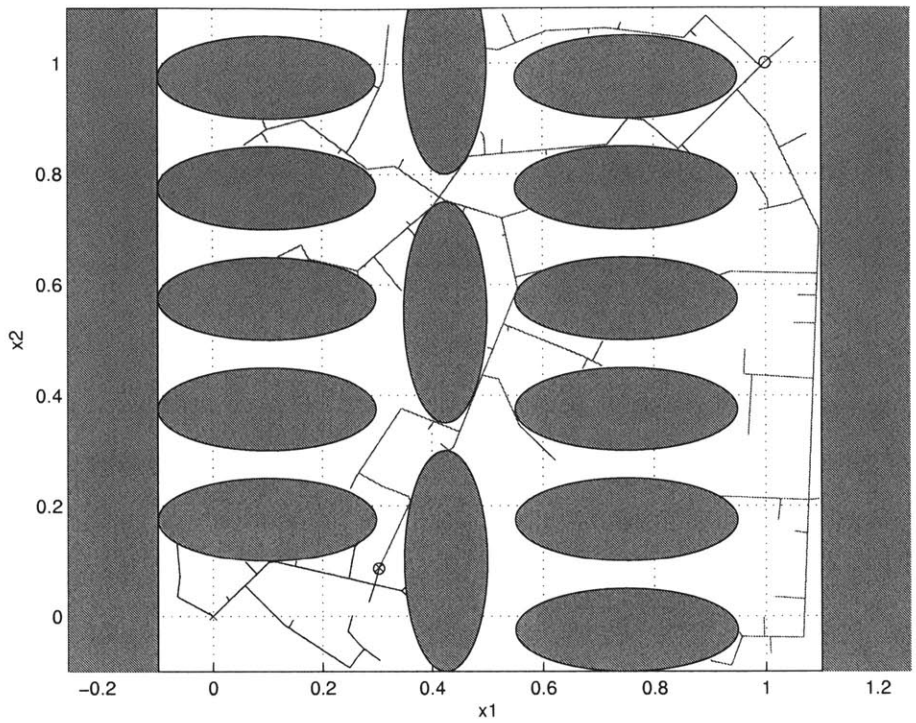
$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2 \tag{3.3}$$

and the ellipsoidal obstacles can be described as the inequalities

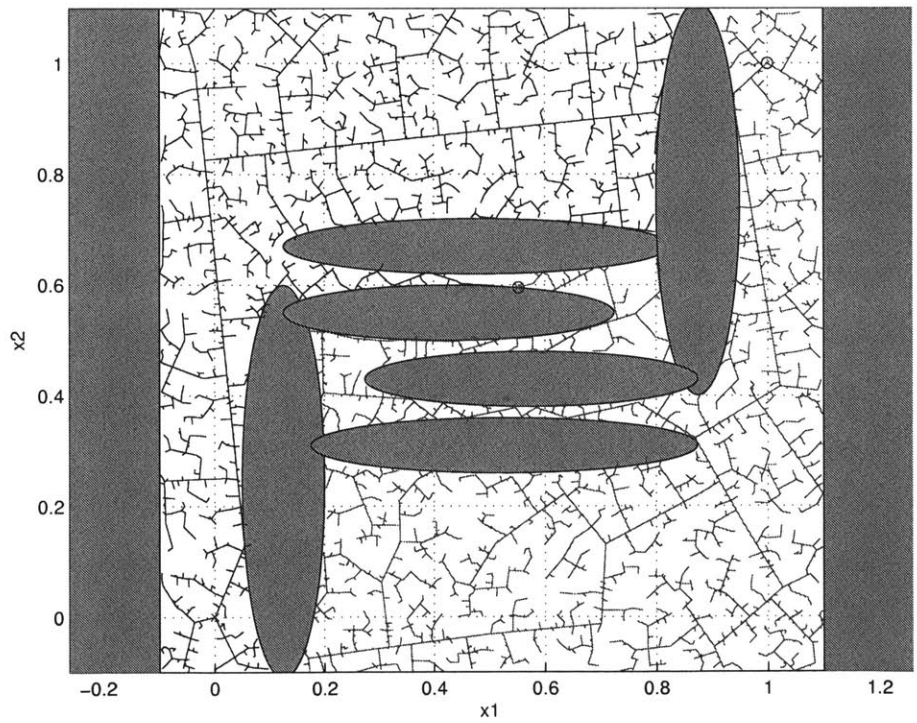
$$1 \leq (x_x - e_{i,x})^2/A_i^2 + (x_y - e_{i,y})^2/B_i^2 \leq \infty \tag{3.4}$$

where $e_{i,x}$ and $e_{i,y}$ are the coordinates of the center of ellipse i, and A_i and B_i the lengths of its x and y axes. Section 3.2.1 explained before that the distance metric and constraint inequalities can be defined separately as long as they are represented like Eqs. 3.1 and 3.2.

The algorithms are evaluated against two specific examples of the problem. Both examples, shown in Figure 3.1(a) and 3.1(b) consist of finding a path from the lower left corner of a square to the upper right corner. The first example, referred to as Example A and shown in Figure 3.1(a), contains scattered obstacles. It is not a trivial example but not particularly difficult, since the passages between obstacles are not very narrow or long compared with the obstacles and the whole space. The second example, referred to as Example B and shown in Figure 3.1(b), is much more difficult.



(a) Example A: Obstacle field



(b) Example B: Narrow passage

Figure 3.1: Simple 2D RRT example with the Direct Connection

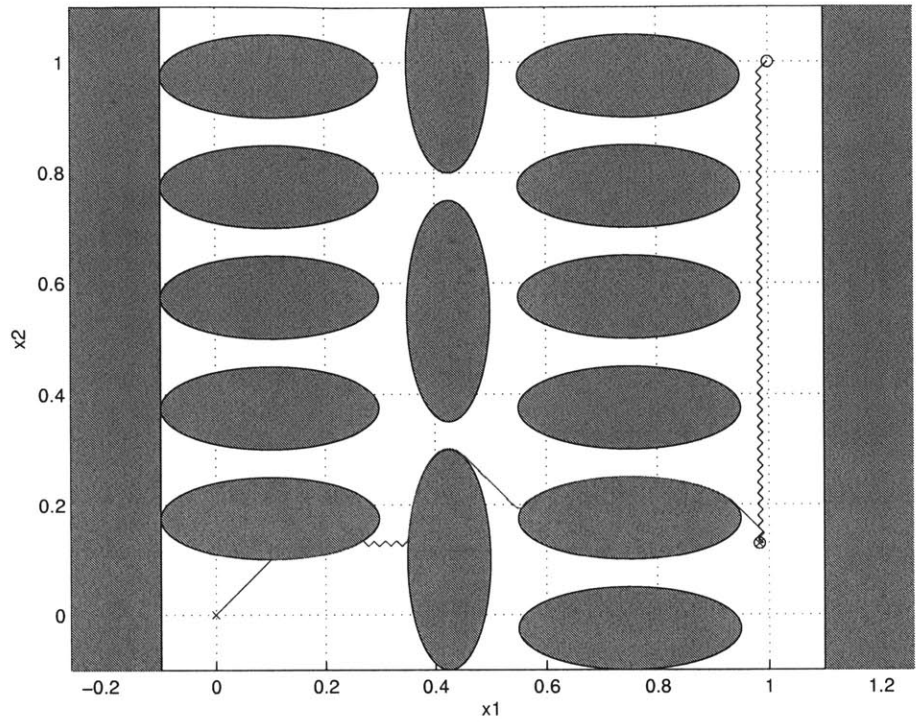
In this example the obstacles are arranged to create a long and narrow passage that must be crossed in order to link the start and goal points. The example was chosen because it is a difficult path planning problem with "narrow passages" and should test the ability of the algorithms to solve problems of this type.

Figures 3.1(a) and 3.1(b) also show the final dual trees created using the direct connection function right up to the point where the trees meet and a path is found. Notice in Figure 3.1(b) that the tree must create several zig-zagging branches to clear the narrow passage. The figure also shows that the basic RRT has performed an exhaustive search and its coverage of the free space is uniform.

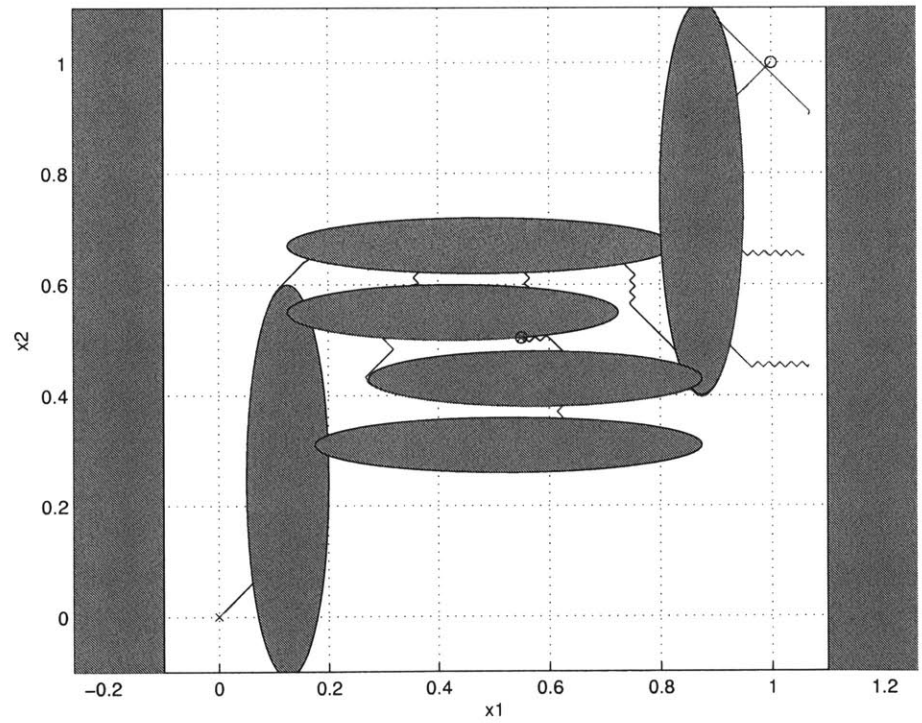
Figures 3.2(a) and 3.2(b) show the bi-directional random trees using the potential connection function. The potential connection function consists of CONNECT using the POTENTIAL-EXTEND function, function d as described in Eq. 3.3 and g constraints as described in Eq. 3.4. The step limit ϵ is of 0.01 units. The figures show that the RRT only needs two branches to solve Example A, which means one iteration, and just a few branches to solve Example B. Figure 3.2(b) also shows that many of branches created by the potential connection function follow the border the obstacles instead of just stopping at them, which is a main characteristic of this connection algorithm.

Figures 3.3(a) and 3.3(b) show the RRT using the randomized connection function for the two sample cases discussed previously, with only one good random trial per step. Besides the random Brownian-motion like shape of the trajectories, the trees bear some resemblance to the normal RRT's. They have multiple branches and explore the free space before connecting the two trees. Since there is only one trial per step, adding the randomization adds only a minimal amount of information. Compare these figures to the trees in Figures 3.4(a) and 3.4(b) with 64 trials per step. These figures look similar to Figures 3.2(a) and 3.2(b), which support the suggestion of section 3.2.2 that a large number of trials provide essentially the same local information as the potential function.

Although these examples highlight the benefits of using local information, such as in the potential function algorithm, it is possible to create examples where this type

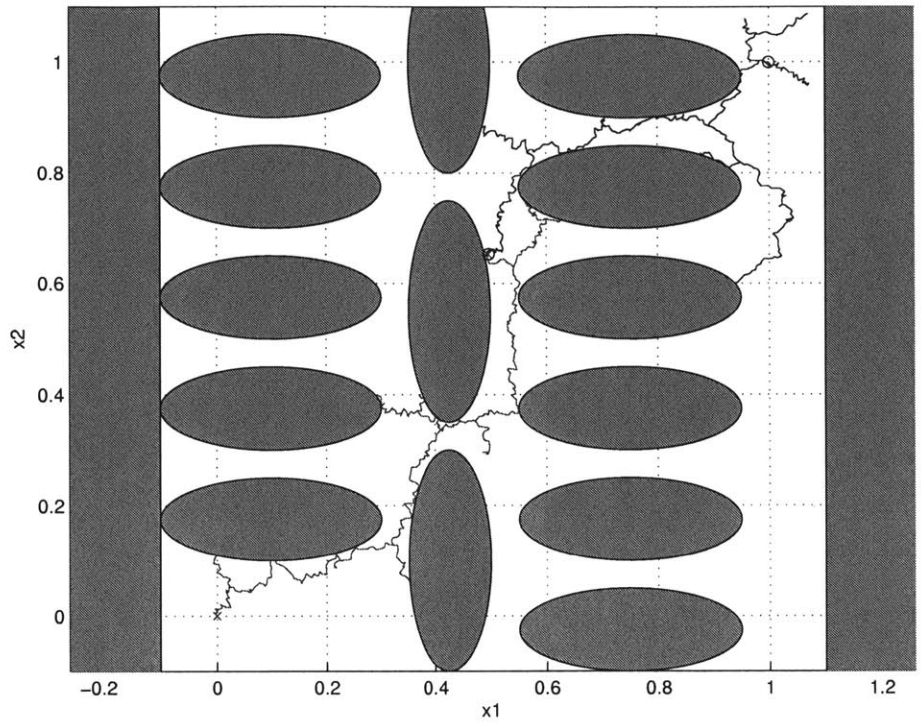


(a) Example A: Obstacle field

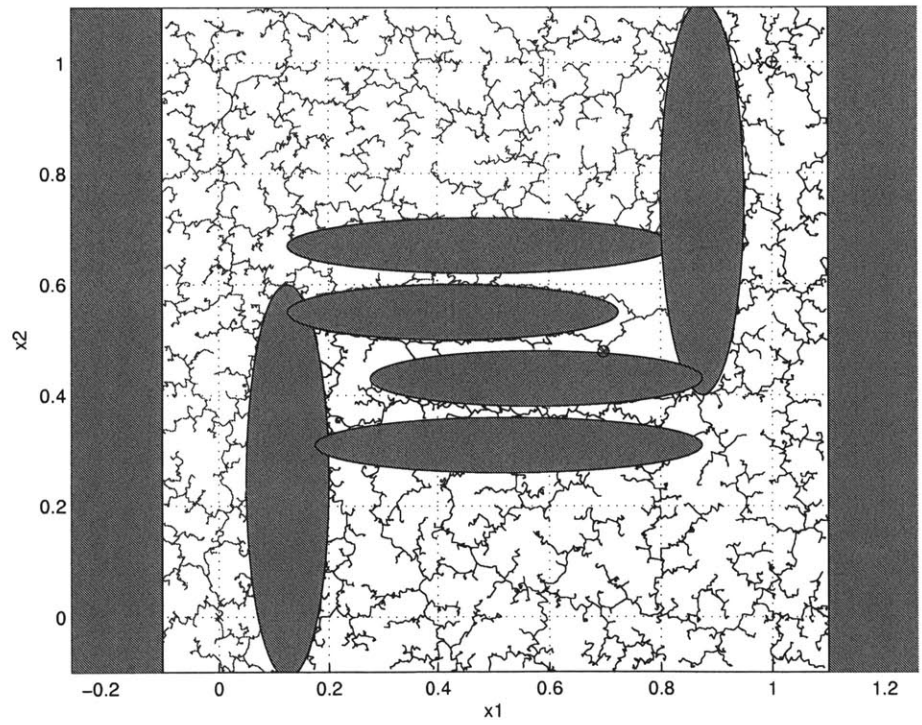


(b) Example B: Narrow passage

Figure 3.2: Simple 2D RRT example with the New Connection

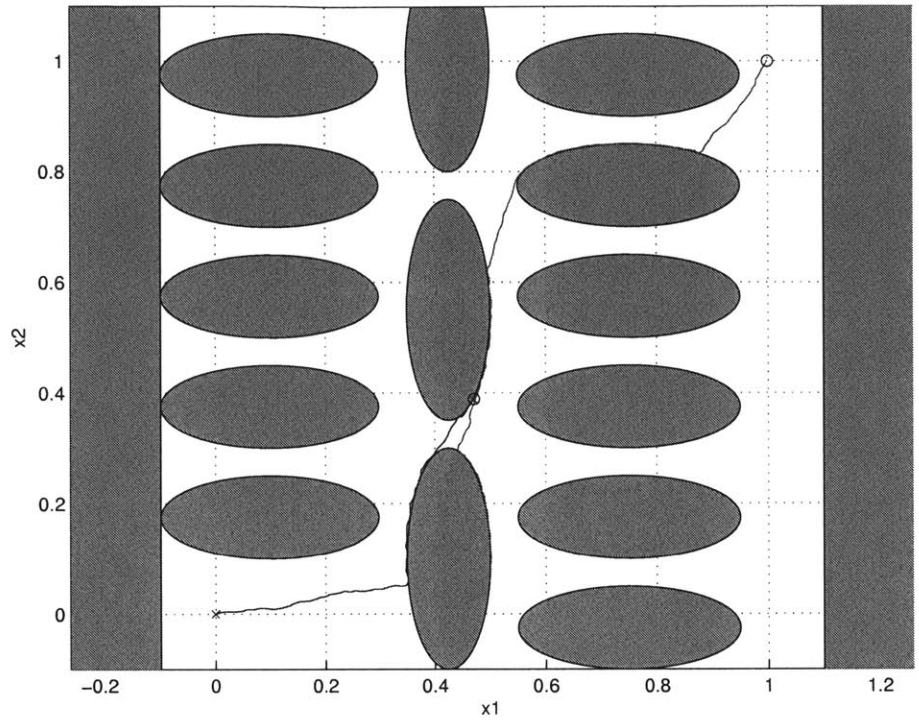


(a) Example A: Obstacle field

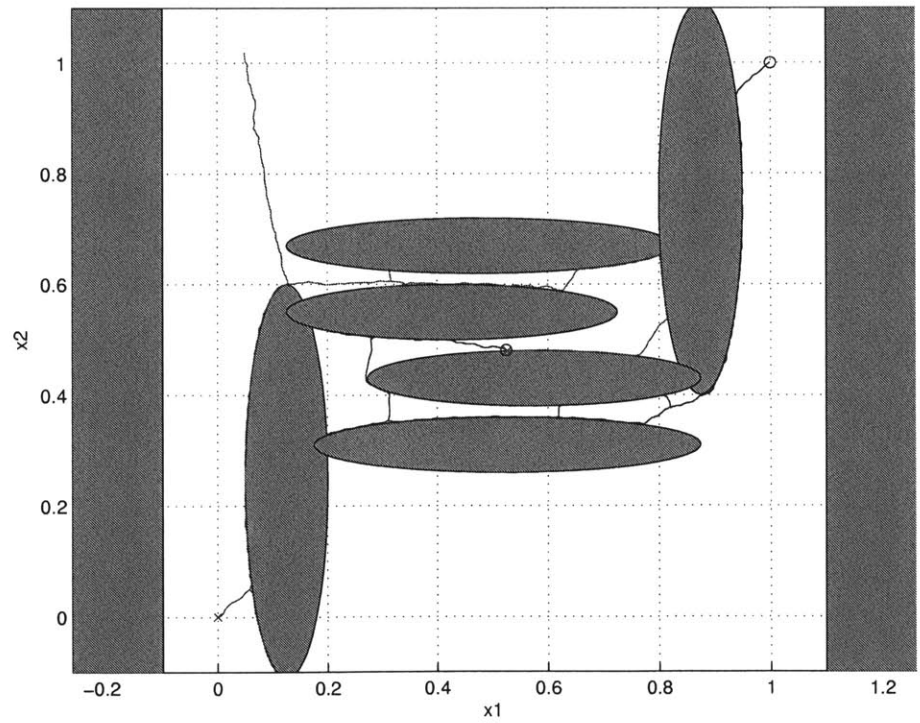


(b) Example B: Narrow passage

Figure 3.3: Randomized connection function with 1 trial per step



(a) Example A: Obstacle field



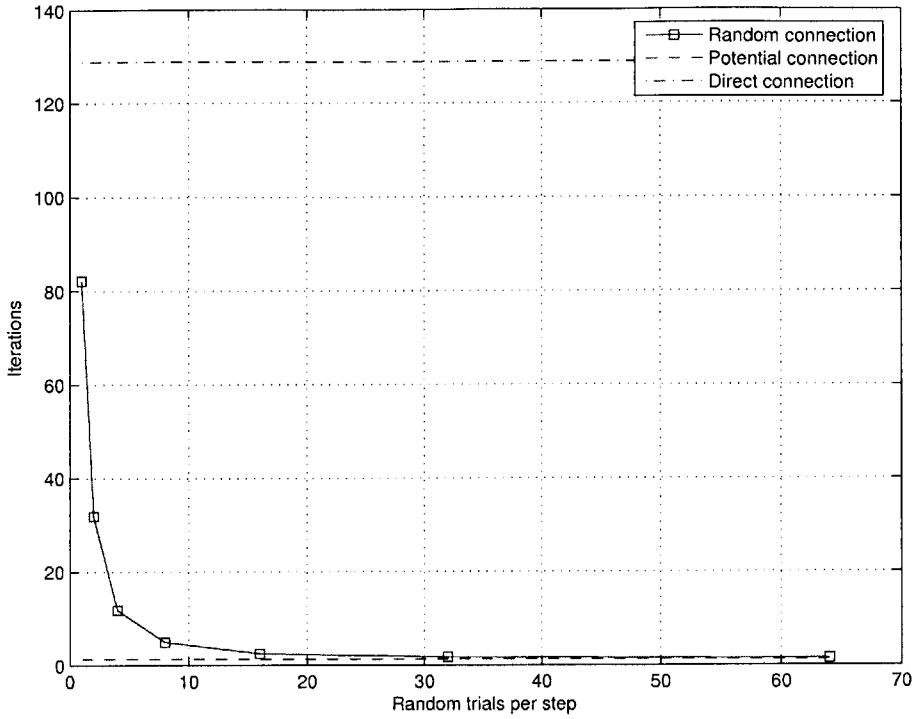
(b) Example B: Narrow passage

Figure 3.4: Randomized connection function with 64 trials per step

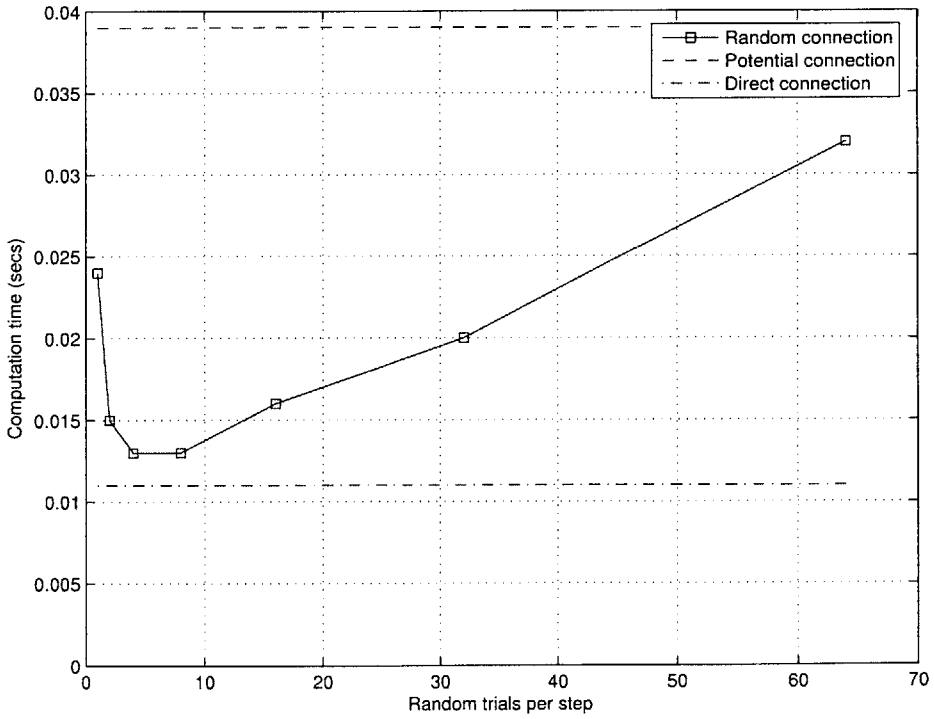
of function does not perform as well as shown here. For instance, if a goal point is at one side of a large obstacle, a potential function trying to reach it from the opposite side of the obstacle will get trapped at local minima. This "trapping" is shown later in the coverage Figures 3.11 and 3.13 in section 3.3.1. Still, the direct connection function has no way of going around the obstacle either, so the potential connection algorithm will never perform worse.

Figures 3.5(a) and 3.5(b) show the average number of iterations of the different RRT's versus the number of random trials per step of the random connection function. The plots show that the number of iterations with the random connection function is asymptotic to the potential connection function, and that the number of iterations with the direct connection function is much larger. This result is consistent with the goal of introducing the two new connection functions: adding information to the connection helps the RRT explore the space faster and therefore reduce the number of iterations to find a solution. However, there is a penalty. The additional complexity of the new connection functions makes each iteration slower. As a result, the new connection functions are slower overall than the basic RRT and the potential connection function is the slowest (Figure 3.5(b)). This result clearly illustrates the reasoning behind the simple connection function in the basic RRT: to have many iterations, but to do each one very fast. These two plots suggest that, at least for this example A, the additional complexity in the functions is not worth it.

Figures 3.6(a) and 3.6(b) show the distribution of computation time of the direct connection function and the potential connection function for Example A, accumulated over 1000 trials and with a maximum cut-off time of 0.8 seconds. These plots show that most of the runs are concentrated below 0.1 seconds, which demonstrate that both algorithms perform well and that the potential connection function does not have a clear advantage for this example. For comparison, Figure 3.6(c) shows the distribution of the random connection function with 8 trials per iteration. This number of trials was shown in Figure 3.5(b) to have the lowest computation time, and this distribution is not different from the plots of the direct function and the potential connection functions, even slightly better.

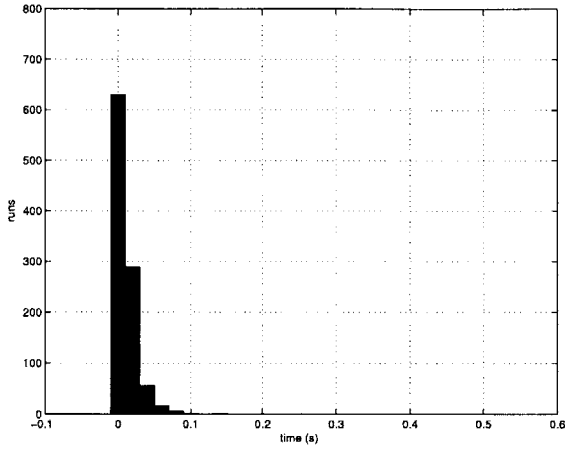


(a) Iterations for Ex. A (average over 1000 runs)

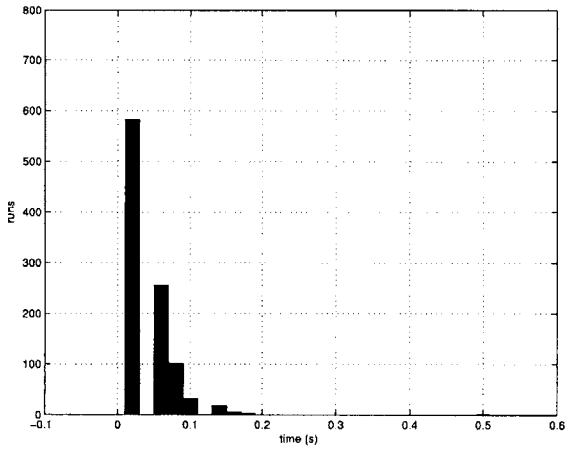


(b) Computation time of search, for Ex. A (avg. over 1000 runs)

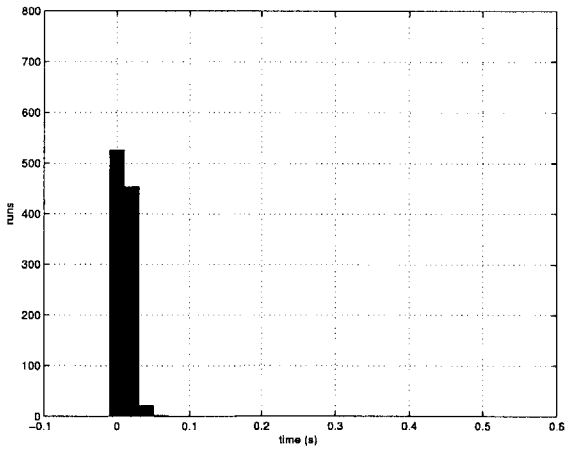
Figure 3.5: Analysis of connection functions for Example A



(a) Direct connection function

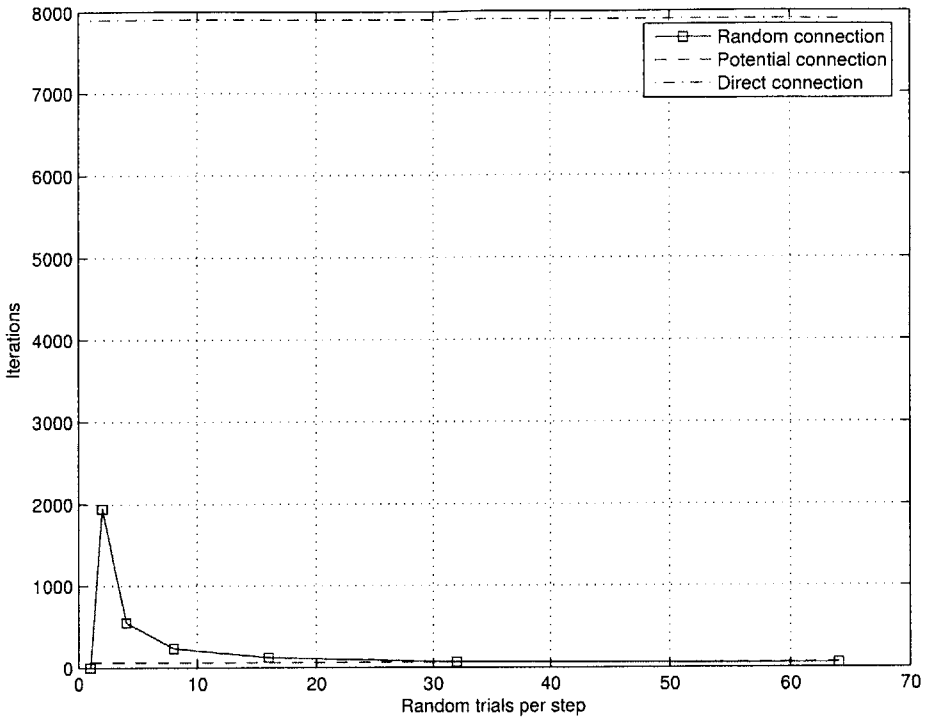


(b) Potential connection function

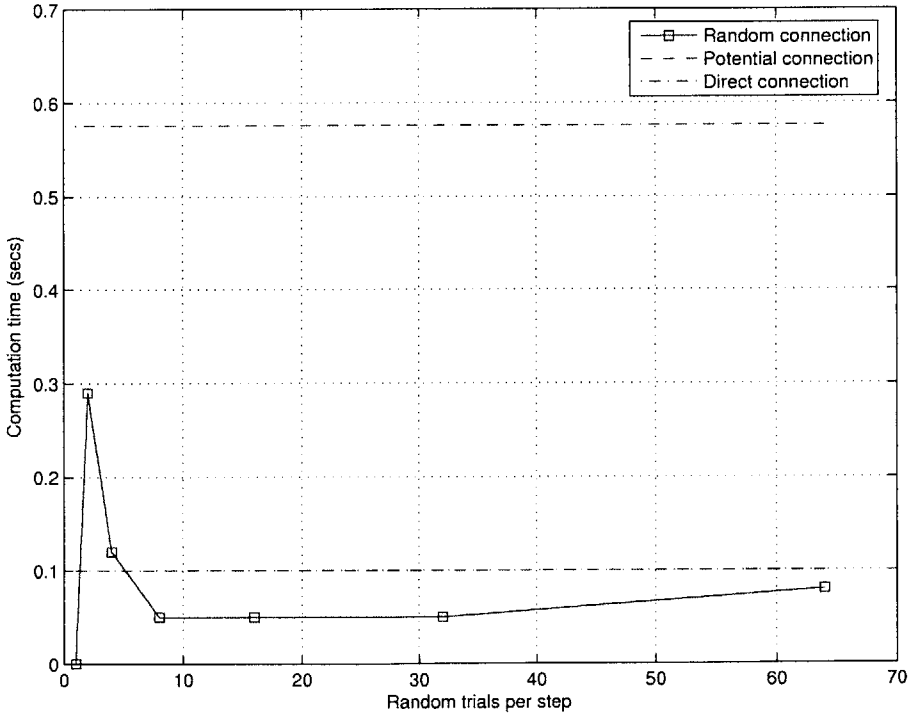


(c) Random connection function with 8 trials per iteration

Figure 3.6: Distribution of computation time for Example A



(a) Iterations for Ex. B (avg. over 1000 runs)



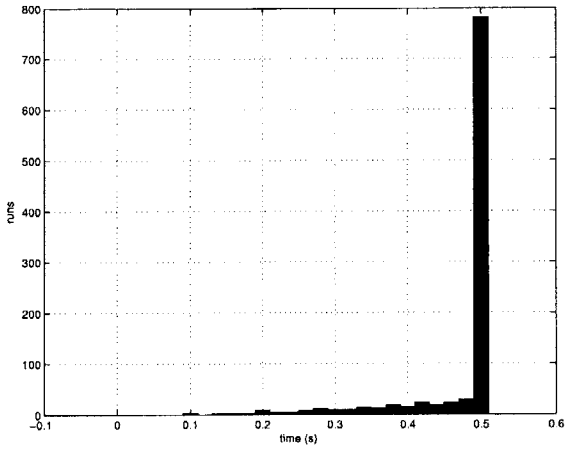
(b) Computation time of search, for Ex. B (avg. over 1000 runs)

Figure 3.7: Analysis of connection functions for Example B

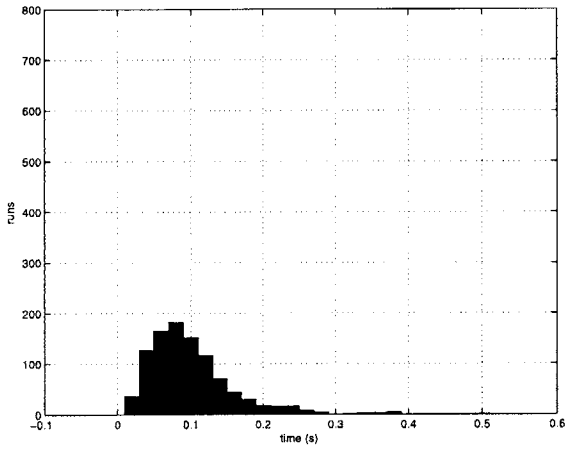
Figures 3.7(a) and 3.7(b) show the iterations and computation time for Example B. They show a similar profile as Figures 3.5(a) and 3.5(b) for Example A for the new connection functions, but very different results for the direct connection function. For the random connection function the computation time is slow for a low number of trials and gets slow also for larger number of trials, while for a mid-range from 8 to 32 trials it is as fast as the RRT with the direct connection function. These figures also show clearly that the computation time of the direct connection function really increases for a hard problem like Example B. Figure 3.7(a) shows that the average number of iterations for the direct connection function has increased to around 7900. As a result, the average computation time increases to nearly 0.6 seconds, even when the time per single iteration may be lower than with the new connection functions. This result confirms that for hard problems the increase in complexity of the connection function does have a positive impact on the performance of the RRT algorithm.

Figures 3.8(a) and 3.8(b) show the distributions of computation time for the direct connection and the potential connection functions. These figures show that the direct connection function has both a worse average time than the potential connection and a wider distribution, which is mostly above 5 seconds. Since randomized algorithms use a cut-off time to decide if the algorithm has failed to find a solution, a wide distribution of computation times has a negative impact in how many runs will fail by stopping too soon, which makes the algorithm less reliable. Figure 3.8(b) shows the computation time of the potential connection function. Although the distribution in this figure has widened compared to Example A (Figure 3.8(a)), it is still concentrated around 0.1 seconds. Figure 3.8(c) shows the distribution of the random connection function with 8 trials per iteration, which was shown to have the best average computation time (Figure 3.7(b)). The randomized connection function has a distribution similar to the potential connection function.

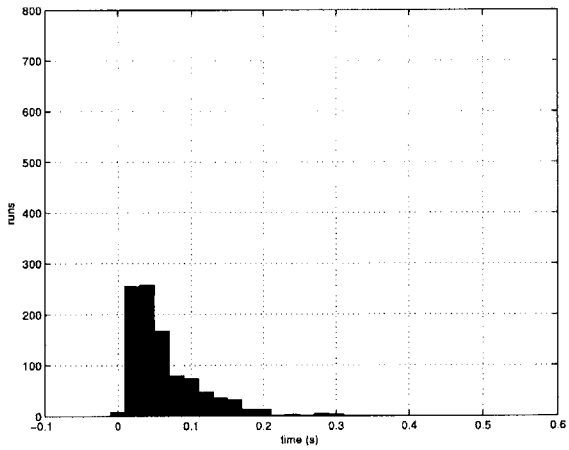
Figures 3.9(a) and 3.9(b) show the average path length for the different algorithms. The plots show a similar behavior to the iteration plots, in the sense that the average path length of the random connection converges asymptotically to a lower bound.



(a) Direct connection function. Times above 0.5 s are counted as 0.5

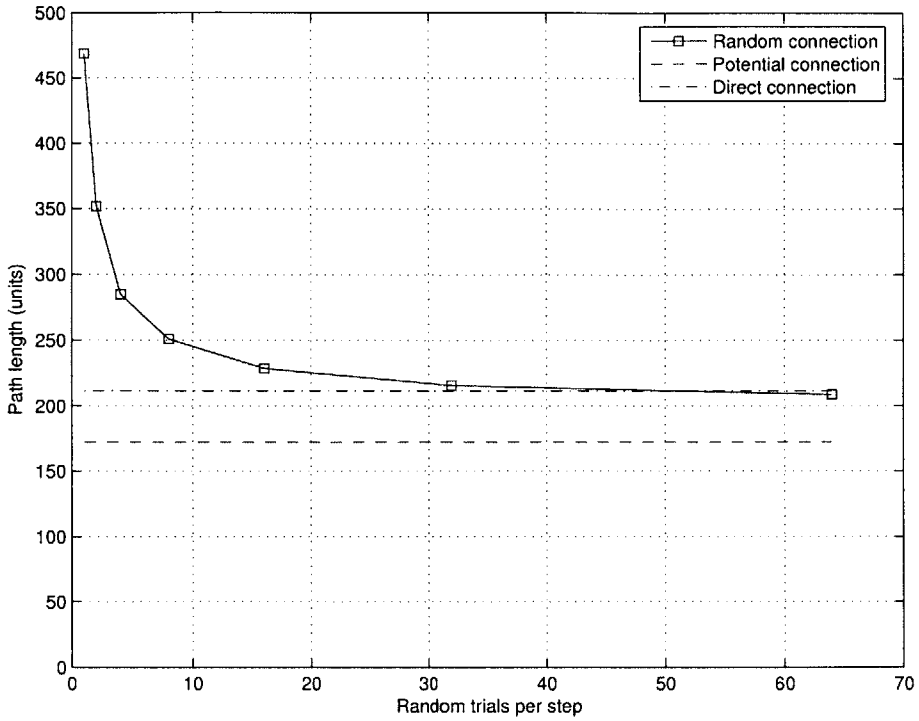


(b) Potential connection function

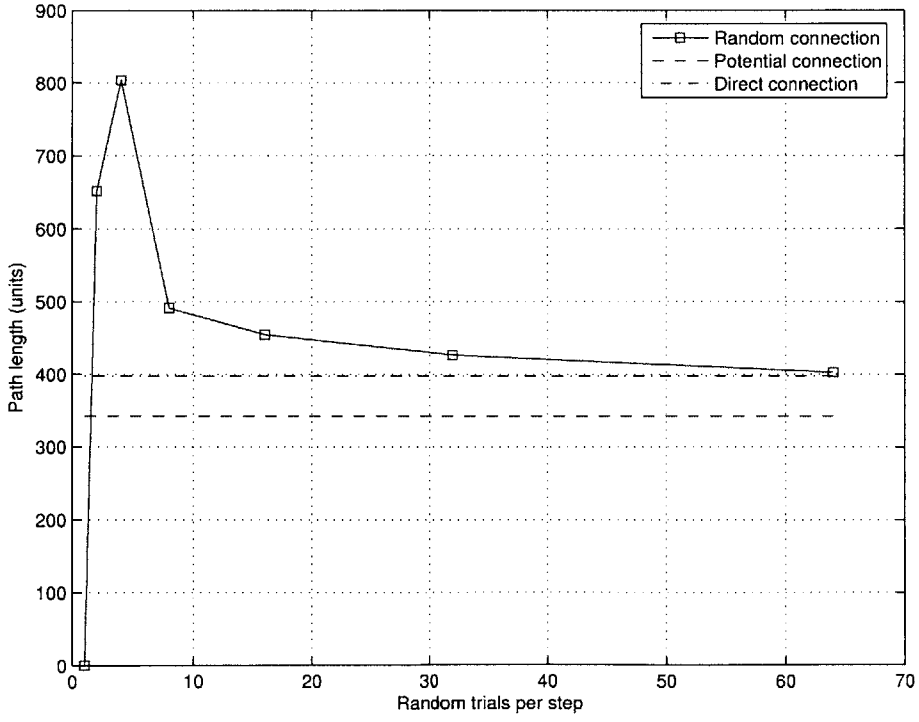


(c) Random connection function with 8 trials per iteration

Figure 3.8: Distribution of computation time for Example B



(a) Path length for Ex. A (avg. over 1000 runs)



(b) Path length for Ex. B (avg. over 1000 runs)

Figure 3.9: Average path length with the different connection functions

Interestingly, the lower bound in this case is consistently equal to the path length of the direct connection RRT. It is not clear why this is the case. The average path length for the potential connection algorithm is consistently lower than the others in these examples. This gives the potential connection algorithm an advantage in path planning problems where the final solution should reduce a cost proportional to the length of the path, for example the time. Although the random connection function has lower computation times, the following examples as well as the spacecraft reconfiguration examples use the potential connection function. This algorithm was chosen because it is an extreme case. It always uses the least number of iterations, and it is deterministic, so it requires no adjustments while the random function ideal number of trials depends on the problem. The potential connection function is also a good reference in terms of connection algorithms. The other extreme case is the direct connection, for which no information is gathered at all, which is evaluated in extensive RRT research and in Chapter 2. The random approach falls between these two algorithms and can be adjusted from one extreme to the other. It is also non-deterministic and not as novel as the potential connection function in the context of randomized algorithms.

3.3.1 Comparison of Coverage

The new potential and random connection functions are presented in section 3.2 as alternatives to the direct connection function in the RRT algorithm. They are then shown to increase the speed and reliability of the RRT algorithm in section 3.3. Figures 3.2(a), 3.2(b), 3.4(a) and 3.4(b) illustrated some characteristics of the new connection functions, like how they follow the obstacle outlines but continue past them, reaching much farther than the direct connection. In this section, Figures 3.10 to 3.13 compare the coverage of the direct and the potential connection functions for the Example A, and Figure 3.10 and 3.11 show their coverage for Example B.

In these figures showing the coverage, the shaded regions are those that can be reached from the origin or the goal points in one step of the connection function. Regions of light shade are reachable from one of the two points, while regions with dark

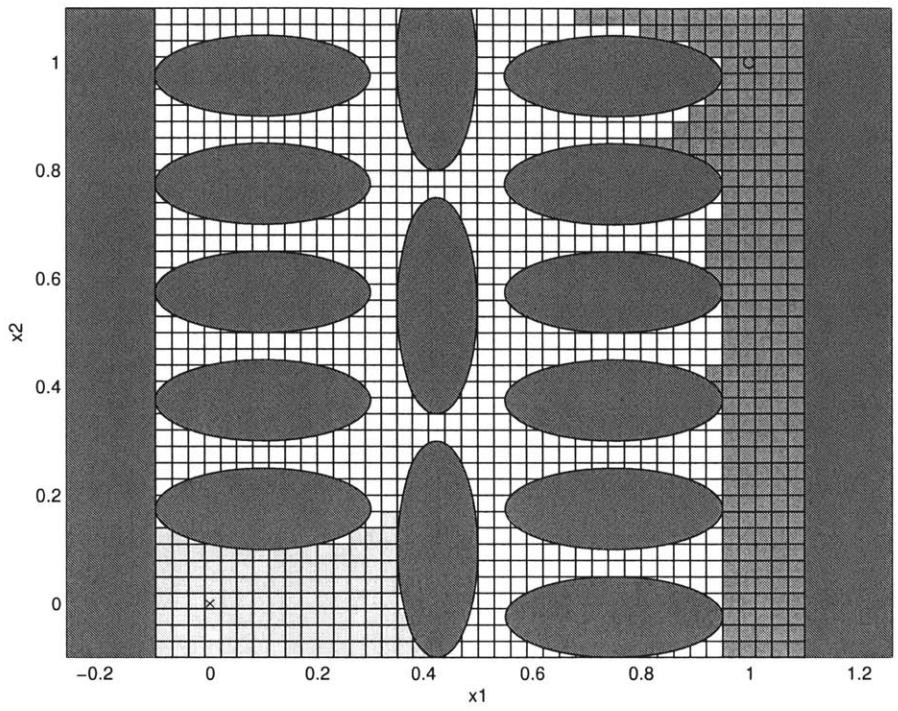


Figure 3.10: Coverage of Ex. A with direct connection. The shaded areas are bounded by the first obstacles

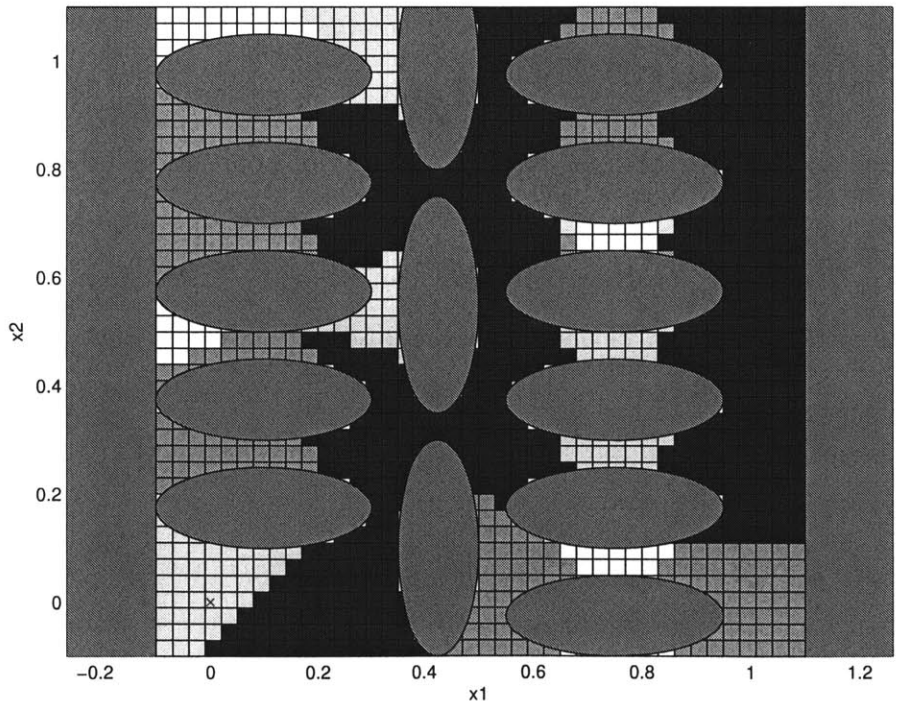


Figure 3.11: Coverage of Ex. A with potential connection. The shaded areas cover most of the free space and overlap in more than half the space (dark). Some regions behind large obstacles are not reachable

shade are those that can be reached from both. Figure 3.10 shows the very limited coverage of the direct connection function. Compare this figure with Figure 3.11 which shows that most of the free space can be covered by the potential connection in one step and that coverage from the origin and goal points overlaps. This explains why the potential connection function finishes most of the searches after one iteration.

Figures 3.12 and 3.13 show similar plots for the narrow passage example (Ex. B). Figure 3.12 shows the limitations of the direct connection function, which can only reach in one step the region within a narrow field of view from the start and goal points. Figure 3.13 shows that the shaded regions have increased, and they also reach deep into the narrow passage in only 1 step. The figure also shows two large regions that are not reachable. These two regions result from the difficulty mentioned in section 3.3, when the potential connection function is trying to reach these areas from the start or goal points but is stuck in local minima behind the large ellipses.

3.4 Simulation Results with the Spacecraft Reconfiguration Problem

In this section the path planner with the potential connection function is tested against the spacecraft reconfiguration problem introduced in Chapter 2. This problem is very challenging problem and has many degrees of freedom. Some of the constraints in this problem, in particular the absolute and relative stay-inside pointing constraints, create difficult regions in the free space that are similar to the "narrow passage" example in section 3.3. The potential connection function has a significant advantage over the basic RRT in problems like these. Other constraints, like the stay-outside pointing constraints and the collision avoidance constraints, are also similar to the obstacles in the Example A (obstacle field). For these cases the potential connection decreases the number of iterations by continuing along the border of the obstacles instead of stopping, as shown in the figures and plots of section 3.3.

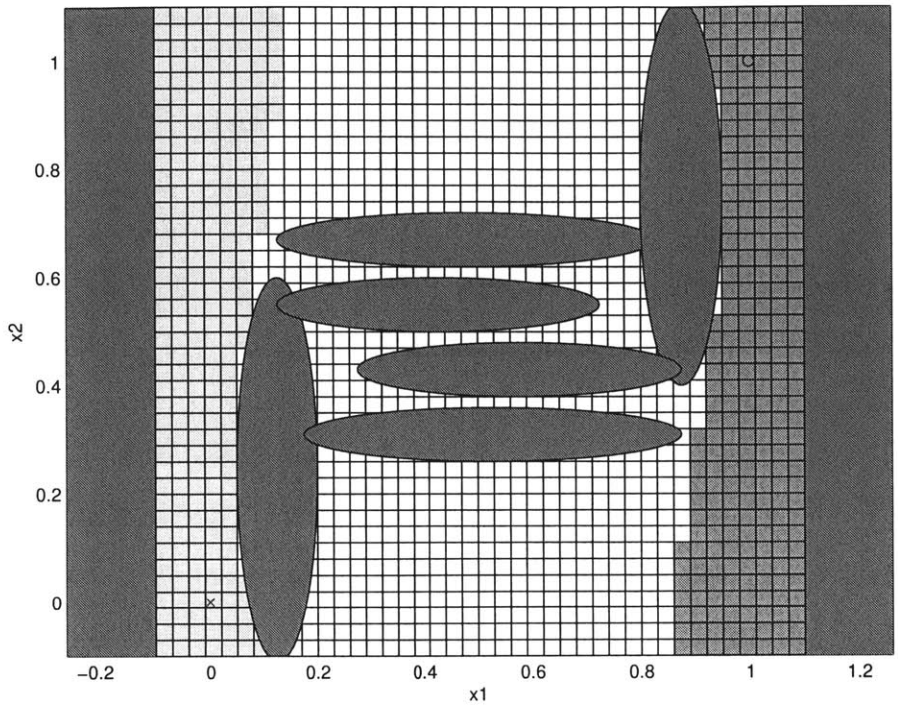


Figure 3.12: Coverage of Ex. B with direct connection. The narrow shaded regions are limited by the limited direct field of view from the start and goal

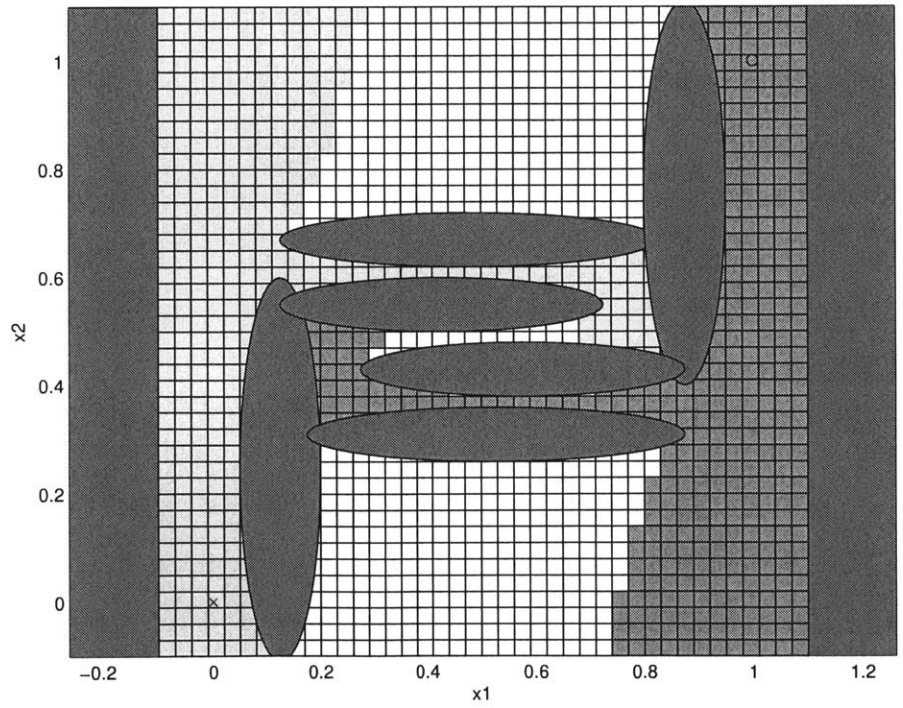


Figure 3.13: Coverage of Ex. B with potential connection. The shaded regions have increased, although limited by local minima in the potential function. The shaded regions also reach deep into the narrow passage

For the spacecraft reconfiguration problem, the distance metric is

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| + K\angle(\boldsymbol{q}_1, \boldsymbol{q}_2) \tag{3.5}$$

where $K = 0.1$ is a scaling factor, and with the collision avoidance and pointing constraints described by Eqs. 2.10 to 2.15.

This section presents examples of different complexity. The first example illustrates the difference between the direct connection function versus the new connection. The next set of the examples compare the computation times of the new connection connection function against results from Chapter 2 with 3 and 4 spacecraft and relative pointing constraints. They are followed by other examples demonstrate the capacity of the improved algorithm to handle larger spacecraft formations, with 8 to 16 spacecraft.

3.4.1 Three Spacecraft

This example has three spacecraft with coupling of attitudes and positions. Spacecraft 1 and 2 switch positions while pointing their X body axis at each other. Meanwhile spacecraft 3 must point the X body axis at them. Additionally, the spacecraft cannot point the X body axis toward the Sun. This constraint is intended to force spacecraft 1 and 2 to move on a plane tilted with respect to the inertial Sun vector.

3.4.2 Change of Formation Shape

In this example four spacecraft begin in a square formation, and they must finish in a tetrahedral position. The constraints are as follows: (a) Spacecraft 1, 2 and 3 must point their Y body axis toward spacecraft 4; and (b) they must also point their X body axis to the next spacecraft in the sequence (spacecraft 1 to spacecraft 2, 2 to 3, and 3 to 1). These 6 constraints place strong restrictions on the possible movements of the spacecraft, which must all be closely coordinated. The attitude of spacecraft 4 is not constrained.

3.4.3 Formation Rotation with 4 Spacecraft

This example consists of a rotation of a tetrahedral formation of four spacecraft. The configuration is the same as described in the previous example, with the same relative pointing constraints, and the formation must rotate 90° about the inertial Y axis.

3.4.4 Formation Reflection with 4 Spacecraft

This example begins with a tetrahedral configuration as well, and the same relative pointing constraints as in the previous two sections. In this example spacecraft 4 (the off-plane spacecraft) is required to cross the plane of the other three spacecraft and switch to the other side.

3.4.5 Comparison of Computation Time

Table 3.1 shows that with the change in the connect function, the path planner solves the problems more than 100 times faster. These problems were carefully chosen for Chapter 2 because they were difficult and representative of real spacecraft reconfiguration maneuvers, but they are easily solved by the new algorithm.

Table 3.1: Comparison with previous results for small examples.

Example	Previous Approach		New Approach	
	Iterations	Time (s)	Iterations	Time (s)
3 s/c (3.4.1)	336	652	2	3
4 s/c shape change (3.4.2)	386	902	1	3
4 s/c rotation (3.4.3)	51	27	2	2
4 s/c reflection (3.4.4)	309	492	1	3

3.4.6 New Results with Larger Configurations

The following examples are shown with configurations of eight and sixteen spacecraft. They include relative pointing constraints that couple the attitude and positions of all spacecraft.

3.4.7 Crossing at the center

In this problem the spacecraft start in the corners of cubes (one for 8 spacecraft, two for 16), facing each other in pairs. They are required to point their X body vector at each other within a 40° angle. The maneuver then consists of switching places with the other spacecraft in the pair. This maneuver is chosen so that all the spacecraft would coincide at the origin if they were to move in straight line. The resulting maneuver is a complex coordinated movement of spacecraft trying to avoid each other while still pointing at their respective pairs and moving at near optimal trajectories. Figures 3.14 and 3.15 show the trajectories for 8 and 16 spacecraft. These trajectories are shown after they have been through some smoothing, since the original trajectories produced by the planner are difficult to read.

3.4.8 Rotation of Star Configuration

This problem consists of a rotation of 90° of a star-like configuration: a spacecraft in the center and the remainder in concentric rings (two rings for eight spacecraft, three rings for sixteen spacecraft) The spacecraft must always point the X body axis (within 40°) toward the spacecraft in the rim that follows counter-clockwise. It should also point a body vector toward the spacecraft in the center (again within 40°). The choice of which vector depends on the size of the formation, but it is fixed during the maneuver.

3.4.9 Random Diagonal Configuration to Star Configuration

In this problems the spacecraft start close to a diagonal configuration pointing the X body axis at the spacecraft that is next in the star configuration (see example 3.4.8). To remove the symmetry in the example, the positions were perturbed with small random displacements, and the attitudes were perturbed by choosing a random rotation axis and rotating the spacecraft a small random angle. If the perturbed configuration satisfy all the constraints then it is accepted, otherwise the process is repeated with the initial configuration until a valid perturbed configuration is found. The spacecraft

must then move to the the star-like configuration described in the example 3.4.8.

Table 3.2: New results for the larger configurations

Problem	# Spacecraft	Iterations	Time (sec)
Crossing at Center (3.4.7)	8	3	29
Rotation of Star (3.4.8)	8	3	9
Diagonal to Star (3.4.9)	8	1	17
Crossing at Center (3.4.7)	16	1	307
Rotation of Star (3.4.8)	16	23	349
Diagonal to Star (3.4.9)	16	3	388

3.5 Summary

This chapter introduces two extensions to the basic Rapidly-exploring Random Tree algorithm that greatly decrease its computation time. These two extensions are based on the notion that path planning problems with constraints have a global scale and a local scale, and that although the basic RRT is very good at searching at the global scale, improvements can be made at the local scale. The extensions consist of replacing the basic RRT function that connects two point, consisting of a simple direct connection, by a function that uses some local information about the constraints. These new functions are better at connecting two points in difficult problems, and as a result improve the overall performance of the whole RRT. One of the functions is based on probing the local free space with a number of random trials, and the other is based on artificial potential functions.

These algorithms and the basic RRT are illustrated and compared with a basic 2D example. The comparison shows that for a simple example the new connection functions find the solution with less iterations than the original function, but not faster. However, for a difficult example the new algorithms perform much better than the basic one in terms of reliability and computation time. The basic RRT is also compared to the potential function connection by solving the spacecraft reconfiguration problem. In potential function algorithm is more reliable and 100 times faster than the basic one. The algorithm is then demonstrated by solving large problems involving up to 16 spacecraft (96 DOF) and many constraints.

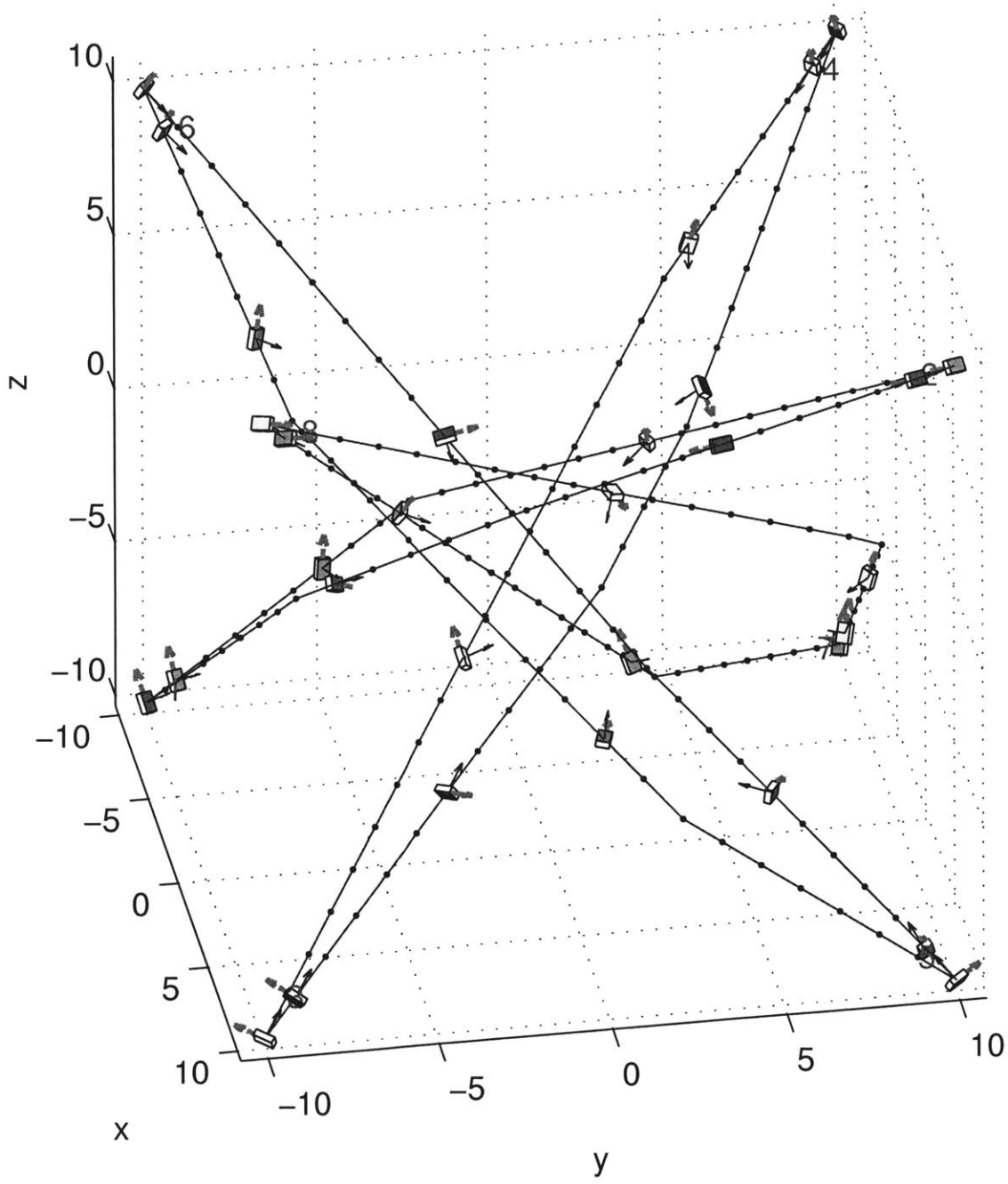


Figure 3.14: 8 spacecraft start at the corners of a cube and switch places

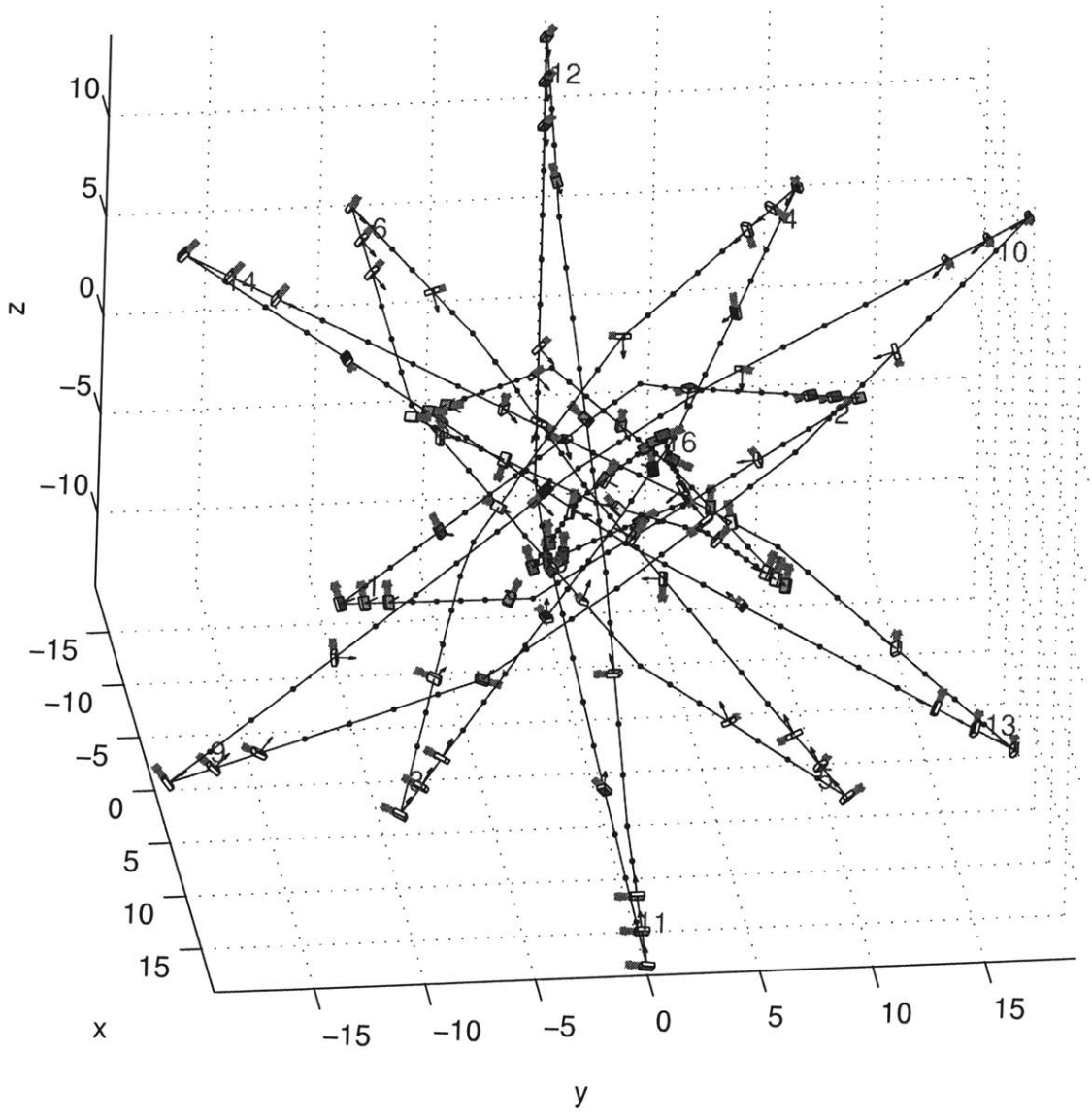


Figure 3.15: 16 spacecraft start at the corners of 2 cubes and switch places

Chapter 4

DIDO and NTG Applied to Trajectories with Discontinuous Constraints

4.1 Introduction

Previous chapters deal with the spacecraft reconfiguration problem. Some of the main features of this problem are its collision avoidance and pointing constraints. These are good examples of path constraints, which must hold for every point of the trajectory. However, there are trajectory optimization problems with constraints that must hold only in certain regions of the space, in certain section of the trajectory, or during a time interval. In other cases the constraints may hold over the hold trajectory, but change sharply in some sections. This sharp changes can apply also to the cost of the trajectory, like travel time of a vehicles that crosses from traveling over land to traveling over swamp. In all the cases mentioned before the problems have discontinuities in the constraints or in the cost function.

In the original formulation of the optimal control problems addressed by DIDO, NTG and other direct methods, the path constraints are limited to be continuous and must be defined over the whole trajectory. Since there are many practical problems

that have discontinuities, it is useful to have a way of solving this type of problems with direct methods. An approach that has been proposed is to divide the problem in phases that are continuous but can have discontinuous transitions between them [3, 40, 45]. However, Ref. [3] also points out there is no systematic way has been developed to solve discontinuous problems with multi-phase methods, and most direct methods did not include the possibility of handling multiphase problems directly until recently (DIDO) [45, 46].

With the aim of tackling discontinuous problems that arise in trajectory optimization, in particular in recent UAV research, this chapter presents the multi-phase formulation of nonlinear control problems, and introduces a systematic method of transforming this problems with this formulation into continuous problems that can be solved by most direct methods. The objective of this chapter is to identify when a discontinuous trajectory optimization problem is suitable for multi-phase formulation, to explain how to solve this problem with existing methods, in particular DIDO or NTG, and to illustrate this process with two examples that are relevant to UAV trajectory design.

From the two software packages dealt with in this chapter, DIDO is a special case because it already includes the multi-phase formulation and can solve multi-phase problems [45]. Therefore section 4.3, which explains how to transform a multi-phase problem into a continuous one, is not relevant to this software, although it is for NTG and other solvers. The rest of the chapter, including the explanation of the multi-phase formulation, how to apply it to discontinuous problems, and the examples, is relevant to all the direct methods including DIDO.

4.2 The Nonlinear Problem

Consider the following optimal control problem. The objective is to minimize the cost functional

$$J = \varphi\left[\boldsymbol{x}(0), 0, \boldsymbol{x}(T), T\right] + \int_0^T L\left[\boldsymbol{x}(t), \boldsymbol{u}(t), t\right] dt \tag{4.1}$$

subject to the dynamic constraints

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}\left[\boldsymbol{x}(t), \boldsymbol{u}(t), t\right] \tag{4.2}$$

the path constraints on state and control variables

$$g_{i,min} \leq g_i\left[\boldsymbol{x}(t), \boldsymbol{u}(t), t\right] \leq g_{i,max} \tag{4.3}$$

and the initial and terminal constraints

$$\boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{4.4}$$

$$\boldsymbol{x}(T) = \boldsymbol{x}_f \tag{4.5}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the state, $\boldsymbol{u}(t) \in \mathbb{R}^m$ is the control input, $t \in \mathbb{R}$ is the independent variable (usually time), $f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^n$, $g_i : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}$, $\boldsymbol{x}_0 \in \mathbb{R}^n$ and $\boldsymbol{x}_f \in \mathbb{R}^n$.

In NTG and DIDO the functions $\boldsymbol{f}$ and g_i must be continuous, due mostly to limitations in the nonlinear optimization solvers that these two packages use. For example, consider the minimum time control problem with a double integrator. Minimize

$$J = \int_O^T dt \tag{4.6}$$

subject to the dynamic constraints

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(t) \tag{4.7}$$

$$\frac{d\boldsymbol{v}}{dt} = \boldsymbol{u}(t) \tag{4.8}$$

the path constraints on the control variable

$$\|\boldsymbol{u}(t)\| \leq U_{max} \tag{4.9}$$

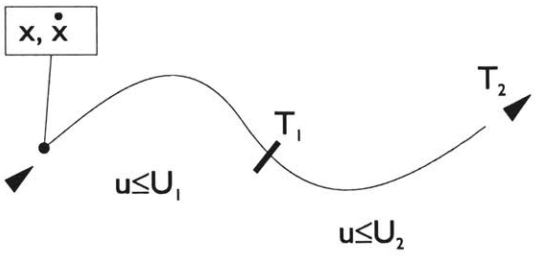


Figure 4.1: Constraint on norm of $\boldsymbol{u}$ changes sharply at $t = T_1$. Final $T_f = T_2$

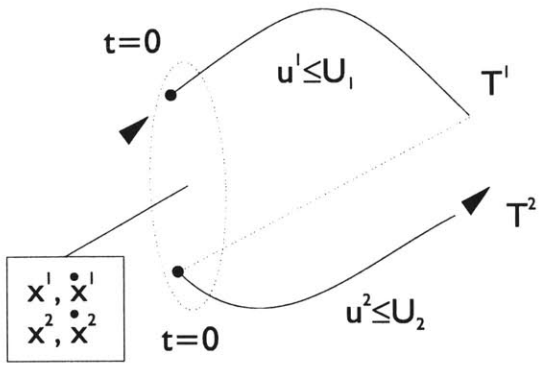


Figure 4.2: Multiphase: Trajectory is split in two phases and then combined. Each phase has a different (but continuous) constraint on the norm of $\boldsymbol{u}$. Final $T_f = T^1 + T^2$

and the initial and final constraints

$$\boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{4.10}$$

$$\boldsymbol{x}(T) = \boldsymbol{x}_f \tag{4.11}$$

However, what happens in the case when these functions are not continuous? In many control problems, the dynamic constraints, path constraints, or the cost may have discontinuities. For example, consider the modification of the previous example in which the bound on the input changes depending on the time

$$\|\boldsymbol{u}(t)\| \leq U_1, \quad t \in (0, T_1) \tag{4.12}$$

$$\|\boldsymbol{u}(t)\| \leq U_2, \quad t \in (T_1, T_2) \tag{4.13}$$

which is shown in Figure 4.1. It should be possible to use either DIDO or NTG to solve problems with discontinuities as well. To achieve this, the focus will narrow to a particular case of discontinuity.

4.3 Problems with Known Path Discontinuities

Consider a trajectory optimization problem divided in M phases, in which each phase can be described similar to section 4.2, but with cost, variables and dynamics that may be different for each phase. A phase $k \in 1, \ldots, M$ can be considered as a trajectory optimization problem in itself, independent of the other phases. The trajectory optimization problem for phase k consists of minimizing

$$J^k = \varphi\left[\boldsymbol{x}^k(0), 0, \boldsymbol{x}^k(T^k), T^k\right] + \int_0^{T^k} L\left[\boldsymbol{x}^k(t), \boldsymbol{u}^k(t), t\right] dt \tag{4.14}$$

subject to the dynamic constraints

$$\frac{d\boldsymbol{x}^k}{dt} = \boldsymbol{f}^k\left[\boldsymbol{x}^k(t), \boldsymbol{u}^k(t), t\right] \tag{4.15}$$

the path constraints

$$g^k_{i,min} \leq g^k_i\left[\boldsymbol{x}^k(t), \boldsymbol{u}^k(t), t\right] \leq g^k_{i,max} \tag{4.16}$$

and the initial and terminal constraints

$$\boldsymbol{x}^k(0) = \boldsymbol{x}^k_0 \tag{4.17}$$

$$\boldsymbol{x}^k(T) = \boldsymbol{x}^k_f \tag{4.18}$$

The independent variable t goes from 0 to T^k for each phase k. The constraints can be different for each phase, therefore $i \in 1, \ldots, N^k$ for phase k. The variables and functions have the same meaning as in equations 4.1 to 4.5. Additionally, the functions L^k, $\boldsymbol{f}^k$ and g^k_i can be different for every phase k. With the optimal control problem for each phase defined, the multi-phase optimal control problem can be defined by adding the objectives and linking the individual trajectories. The new objective is to minimize

$$J = \sum_{k=1}^{M} J^k \tag{4.19}$$

and the linking is achieved by constraining the final and initial states of adjacent trajectories to be equal

$$\boldsymbol{x}^k(T) = \boldsymbol{x}^{k+1}(0) \tag{4.20}$$

$$\boldsymbol{u}^k(T) = \boldsymbol{u}^{k+1}(0) \tag{4.21}$$

where $k \in 1, \ldots, M-1$.

Although this form forces all the states and inputs to be the same at the linking points, it can actually be more flexible. It is possible not to include some of these linking constraints for some of the states (like velocity) or the control inputs. For example, if a particular problem allows the acceleration to change sharply between phases, the constraint on this control input can be relaxed. With some extra work, the linking conditions could be made even more flexible and complex, but the problems considered here do not require this flexibility, so this option is not explored further.

Figure 4.2 shows the multi-phase formulation of the discontinuous example described in section 4.2. The states $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$ and control $\boldsymbol{u}$, have been replaced by two sets of states and controls $\boldsymbol{x}^1$, $\dot{\boldsymbol{x}}^1$, $\boldsymbol{x}^2$, $\dot{\boldsymbol{x}}^2$, $\boldsymbol{u}^1$ and $\boldsymbol{u}^2$, a set per phase. Two different norm constraints are enforced at the different phases, and the continuity between phases of the trajectory is enforced by the linking constraints. The total time is the sum of each phase time $T_f = T^1 + T^2$.

This type of control problem can be formulated in the first form shown in section 4.2 by introducing new variables and constraints. Let the times T^k be dependent variables and introduce a new independent variable $\tau \in [0, 1]$ such that

$$t = \tau T^k \tag{4.22}$$

$$dt = d\tau T^k \tag{4.23}$$

The multiple-phase optimal control problem is then to minimize

$$J = \sum_{k=1}^{M} \varphi^k \left[\boldsymbol{x}^k(0), 0, \boldsymbol{x}(1), T^k\right] + \sum_{k=1}^{M} \int_0^1 T^k L^k \left[\boldsymbol{x}^k(\tau), \boldsymbol{u}^k(\tau), \tau T^k\right] d\tau \tag{4.24}$$

subject to the dynamic constraints

$$\frac{d\boldsymbol{x}^k}{d\tau} = T^k \boldsymbol{f}^k \left[\boldsymbol{x}^k(\tau), \boldsymbol{u}^k(\tau), \tau T^k\right] \tag{4.25}$$

the new dynamic constraints

$$\frac{dT_k}{d\tau} = 0 \tag{4.26}$$

and the path constraints

$$g^k_{i,min} \leq g^k_i \left[\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \tau T^k\right] \leq g^k_{i,max} \tag{4.27}$$

where $k \in 1, \ldots, M$. The initial and terminal constraints are

$$\boldsymbol{x}^1(0) = \boldsymbol{x}_0 \tag{4.28}$$

$$\boldsymbol{x}^M(1) = \boldsymbol{x}_f \tag{4.29}$$

and the linking constraints constraints

$$\boldsymbol{x}^k(1) = \boldsymbol{x}^{k+1}(0) \tag{4.30}$$

$$\boldsymbol{u}^k(1) = \boldsymbol{u}^{k+1}(0) \tag{4.31}$$

where $k \in 1, \ldots, M-1$. Posed in this form, the multi-phase problem is transformed into the original control problem with continuous variables, cost and constraints of section 4.2. The drawback is that the size of the problem increases. The number of variables and inputs has increased from $n+m$ to $M(n+m+1)$ because they are duplicated at each phase, plus each additional T^k per phase. The number of initial and terminal constraints goes from $2n$ to $2n + (M-1)(n+m)$. Finally, the number of path constraints goes from N to MN plus the constraints that are specific to the multi-phase problem.

4.4 Multi-phase Problem in DIDO and NTG

Once the discontinuous problem is transformed into a continuous one, it is straightforward to solve this problem with any solver like DIDO or NTG. With NTG, however, there is an additional limitation due to the software implementation of the version 2.2. In this version of the software it is not possible to include the states and inputs in a constraint at *different times*, as in Eqs. 4.30 and 4.31. This minor limitation is a characteristic of NTG that can be overcome with a simple additional step. It is possible to introduce a variable $\boldsymbol{X}^k$ for each phase $k \in 1, \ldots, M-1$ of the problem, and then transform each linking constraint like Eq. 4.30

$$\boldsymbol{x}^k(1) = \boldsymbol{x}^{k+1}(0)$$

into the equivalent pair of constraints

$$\boldsymbol{x}^k(1) = \boldsymbol{X}^k(1) \tag{4.32}$$

$$\boldsymbol{X}^k(0) = \boldsymbol{x}^{k+1}(0) \tag{4.33}$$

If the value of $\boldsymbol{X}^k$ is made constant over the whole trajectory, it follows that $\boldsymbol{X}^k(0) = \boldsymbol{X}^k(1) = \boldsymbol{X}^k$. The variable $\boldsymbol{X}^k$ is made constant by defining its dynamics as

$$\frac{d\boldsymbol{X}^k}{d\tau} = 0 \tag{4.34}$$

The control inputs are also part of the linking constraints and they should also be included in this step. The new constraints are

$$\boldsymbol{u}^k(1) = \boldsymbol{U}^k \tag{4.35}$$

$$\boldsymbol{U}^k = \boldsymbol{u}^{k+1}(0) \tag{4.36}$$

and the corresponding dynamics are

$$\frac{d\boldsymbol{U}^k}{d\tau} = 0 \tag{4.37}$$

This simple step effectively transforms the multi-phase problem formulated as in section 4.3 into a problem solvable by NTG, at the additional expense of $(M-1)(n+m)$ new constant variables and $(M-1)(n+m)$ more initial and end point constraints. The following sections present two examples of discontinuous problems and use them to compare the performance of NTG and DIDO.

4.5 The Sensor Problem

This problems consists of a variant of the basic 2D Unmanned Aerial Vehicle problem described in Appendix A with additional discontinuous constraints. The basic problem consists of minimum time trajectory design with dynamics modeled as a double integrator with lower limit in speed and a turning radius. As part part of its mission, the UAV is required to point a sensor (*e.g.*, camera) at some points in the map for more than certain time. The basic problem is then formulated as minimizing

$$J = \int_0^T dt \tag{4.38}$$

subject to the dynamic constraints

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(t) \tag{4.39}$$

$$\frac{d\boldsymbol{v}}{dt} = \boldsymbol{u}(t) \tag{4.40}$$

the path constraints

$$V_{min} \leq \|\boldsymbol{v}(t)\| \leq V_{max} \tag{4.41}$$

$$\|\boldsymbol{u}(t)\| \leq U_{max} \tag{4.42}$$

and the initial and final constraints

$$\boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{4.43}$$

$$\boldsymbol{v}(0) = \boldsymbol{v}_0 \tag{4.44}$$

$$\boldsymbol{x}(T) = \boldsymbol{x}_f \tag{4.45}$$

$$\boldsymbol{v}(T) = \boldsymbol{v}_f \tag{4.46}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^2$ and $\boldsymbol{v}(t) \in \mathbb{R}^2$ are the position and velocity of the UAV in 2D, and $\boldsymbol{u}(t) \in \mathbb{R}^2$ is the control input. The turning radius constraints were introduced as the lower bound to the speed of the UAV and the upper bound to the acceleration in Eqs. 4.41 and 4.42. The sensor pointing constraints are introduced in a simplified form for the sake of clarity. For a single point $\boldsymbol{r}_j$, the constraint requires the UAV to fly within a distance R_j of this point for longer than a specified time T_{min}. To state this formally, there must be two times $\mathcal{T}_j$ and $\mathcal{T}_{j+1}$ such that

$$\|\boldsymbol{x}(t) - \boldsymbol{r}_j\| \leq R_j, \quad \mathcal{T}_j \leq t \leq \mathcal{T}_{j+1} \tag{4.47}$$

which must be separated by at least a time T_{min}

$$\mathcal{T}_j + T_{min} \leq \mathcal{T}_{j+1} \tag{4.48}$$

Additionally these two times are restricted to be within the time interval of the problem

$$0 \leq \quad \mathcal{T}_j \quad \leq T \tag{4.49}$$

$$0 \leq \quad \mathcal{T}_{j+1} \quad \leq T \tag{4.50}$$

The general problem may include an arbitrary number of these constraints, and the starting and ending times are not constrained so they may overlap and come in any order.

The problem is quite general as stated before, and it is not possible to actually

solve it with any of the methods introduced previously. The timing constraint of Eq. 4.47 is clearly discontinuous, since it only holds when the time t is between the two variables $\mathcal{T}_j$ and $\mathcal{T}_{j+1}$. This would make the problem a candidate for the multi-phase formulation, but there is an additional limitation.

A key assumption in the multi-phase formulation of section 4.3 is that the start and end times of the discontinuous constraints do not overlap, and that they hold for phases of the trajectory that are predetermined before the optimization. Usually the definition of a problem is enough to determine if the first assumption is valid, and in order to satisfy it a problem may have to be simplified further. The second assumption can be satisfied by using an initial guess to determine how to assign the constraints.

The constraints of Eqs. 4.47 to 4.50 are too general to satisfy the first assumption. Fortunately, the problem can be simplified further and formulated as multi-phase. If the sensing areas are limited not to overlap and an initial guess is used to assign the constraints to the corresponding phases, then the sensor problem can then be formulated as minimizing

$$J = \sum_{k=1}^{M} \int_0^{T^k} dt \tag{4.51}$$

with the dynamics, path and endpoint constraints of Eqs. 4.25 to 4.31. The trajectory should also satisfy the path constraints related to the sensor constraints for those phases k inside sensing regions

$$\|\boldsymbol{x}^k(t) - \boldsymbol{r}^k\| \leq R^k \tag{4.52}$$

$$T_{min} \leq T^k \tag{4.53}$$

where $\boldsymbol{r}_j$ and R_j are the position and distance to the point j that the UAV must sense, and T_j is the time for which this constraint holds (Fig 4.3). With the sensor problem posed as a multi-phase formulation, it can now be solved by DIDO or NTG.

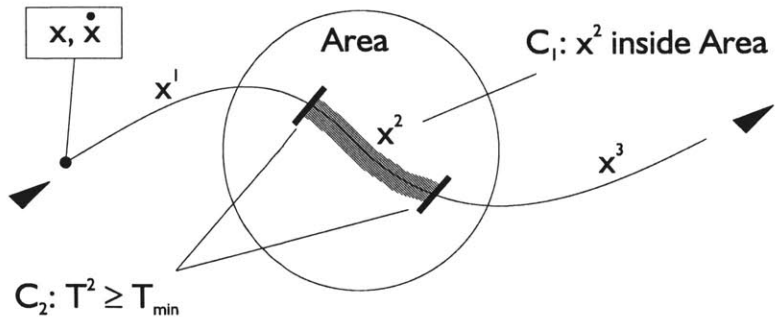


Figure 4.3: The UAV is required to stay within the sensing area for time T_{min} by enforcing constraints like Eqs. 4.52 and 4.53 in phase 2

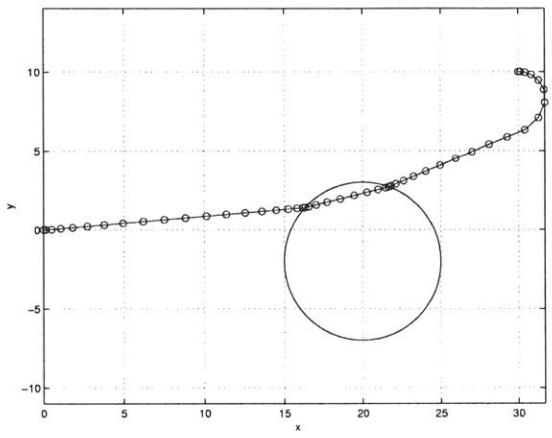


Figure 4.4: Trajectory generated by DIDO for the sensor problem

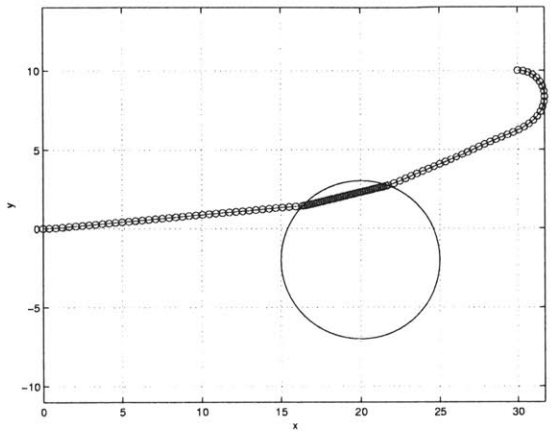


Figure 4.5: Trajectory generated by NTG for the sensor problem

4.6 Simulation of the Sensor Problem

This section compares the performance of DIDO and NTG using a simple example. In this example the UAV must go from point $\boldsymbol{x}_0 = [0,0]^T$,$\boldsymbol{v}_0 = [1,0]^T$ to point $\boldsymbol{x}_f = [10,30]^T$,$\boldsymbol{v}_f = [-1,0]^T$. There is also a sensing point at $\boldsymbol{r}_1 = [20,-2]^T$ with sensing radius of $R_1 = 5$. The norm of the velocity is bounded between $V_{min} = 0.9$ and $V_{max} = 1.1$ and the upper bound on norm of the control input is $U_{max} = 0.5$. The formulation of this problem involves a trajectory with 3 phases: initial point to sensing area, inside the sensing area, and from sensing area to the end point. The 2nd phase is constrained to be inside the sensing area for a time larger than $T_{min} = 9$ with two constraints like Eqs. 4.52 and 4.53.

Figure 4.4 shows the solution produced by DIDO. This solution was found in 91

seconds with 71507 cost function evaluations, and has a cost (in time) of 33.8 seconds. Figure 4.5 shows the solution produced by NTG after 23 seconds and 44100 calls to the cost function. It has a cost (in time) of 34.0 seconds. These results are summarized in Table 4.1. This table shows that DIDO's optimal cost was lower than NTG but not by much. In contrast, it had worst results in terms of computation time, in part because DIDO runs in MATLAB which is not as fast as NTG's compiled code, but also because it seems to have a larger number of iterations. Since DIDO's precision cannot be controlled by the user, it sacrifices speed to decrease the cost as much as possible. Figures 4.4 and 4.5 show very similar trajectories. The only visible differences are the positions of the points that form the trajectory, which correspond to the nodes used in the optimization. In the trajectory by DIDO they are distributed unevenly, which is a characteristic of the Legendre pseudospectral method. The trajectory by NTG shows more nodes and they are distributed evenly, since in NTG the node positions are not key to its performance, and they are usually just distributed evenly over the trajectory. The three phases of the trajectory are also illustrated by the discontinuity of the nodes in Figures 4.4 and 4.5. In DIDO the number of nodes per phase can be adjusted, and it its lower inside the sensing area. In NTG all phases must have the same number of nodes, which is visible in the higher density of nodes inside the sensing area. The figures also show that the constraint of staying in the sensing area for more than 9 seconds forces the trajectory to be suboptimal.

Table 4.1: Results for sensor problem

Software	Cost (T)	Computation time (secs)	Iterations (fn calls)
DIDO	33.8	91	71507
NTG	34.0	23	44100

4.7 The Weighted Shortest Path

This problem also consists of a variant of the basic 2D UAV in which the cost to minimize is not just the time (or distance). In this problem the time has different

weights on the cost depending on the area traveled, that is, the cost is

$$J = \int_0^T W(\boldsymbol{x}(t))dt \tag{4.54}$$

where the weight $W(\boldsymbol{x}(t))$ is an scalar that depends on the position $\boldsymbol{x}(t)$. This problem, or its path planning variant, has been the subject of extensive research [21, 52]. Many control problems that may seem unrelated can be represented in this form, for example, finding the path that poses the minimum risk of losing the UAV, when the $W(\boldsymbol{x})$ represents the probability density of losing the UAV in an area [21].The difficulty is that in most of real life problems the weight function is discontinuous and changes sharply in the boundaries of different areas. In some cases, even if the weights are continuous they change in a very short distance so for practical purposes they are as difficult for the optimization algorithms. However, this problem can be posed in the form of the multi-phase problem in section 4.3. The problem then consists of minimizing

$$J = \sum_{j=1}^{M} \int_{T_{j-1}}^{T_j} W_j(\boldsymbol{x}(t))dt \tag{4.55}$$

with the path constraints

$$\boldsymbol{x}(t) \in \mathcal{A}_j, \quad t \in [T_{j-1}, T_j] \tag{4.56}$$

where $j \in 1, \ldots, M$, W_j is the weight function which is continuous in the area $\mathcal{A}_j$ traversed by the UAV between times T_{j-1} and T_j. This problem formulation is too general and cannot be represented as a multi-phase problem. The limitation is that t is used as a variable in the constraint of Eq. 4.56, but this is not possible in the multi-phase formulation. However, if an initial guess is used to determine the order in which the areas are visited, then it is possible to predetermine the phases of the trajectory and assign the constraints to their corresponding phases. The problem is to minimize

$$J = \sum_{k=1}^{M} \int_0^{T^k} W^k(\boldsymbol{x}^k(t))dt \tag{4.57}$$

with the path constraints

$$g^k\left[\boldsymbol{x}(t)\right] \leq 0, \quad \forall k \in 1, \ldots, M \tag{4.58}$$

and the rest of the dynamic, path and end-point constraints described in section 4.3. Since each weighting function $W^k(\boldsymbol{x})$ corresponds to an area, the corresponding trajectory phase must be constrained to be inside that area by the $g^k(\boldsymbol{x})$ constraints in Eq. 4.58.

The previous formulation can already be solved with NTG and DIDO, however, the formulation of the examples shown here has been further simplified for the sake of clarity. In particular, the weights are limited to be constants and the areas to be convex polygons. Constraining a point to be inside a polygon can be represented with a set of linear functions, one per side of a polygon. The corresponding constraint $g^k(\boldsymbol{x}) \leq 0$ can be represented by the equivalent set of constraints

$$\boldsymbol{v}_j^{k\,T}\left[\boldsymbol{x}^k(t) - \boldsymbol{r}_j^k\right] \geq 0, \quad \forall k \in 1, \ldots, M, \; j \in 1, \ldots, N^k \tag{4.59}$$

where $\boldsymbol{v}_j^k$ is a vector normal to the side j of polygon k, and $\boldsymbol{r}_j^k$ is any point in this side j, as shown in the example of Figure 4.6. The following example compares the performance of DIDO and NTG in the weighted shortest path problem.

4.8 Simulation of the Weighted Shortest Path Problem

In this example the UAV must go from point $\boldsymbol{x}_0 = [0,0]^T$,$\boldsymbol{v}_0 = [1,0]^T$ to point $\boldsymbol{x}_f = [10,30]^T$,$\boldsymbol{v}_f = [-1,0]^T$, as in section 4.6. The norm of the velocity is also bounded between $V_{min} = 0.9$ and $V_{max} = 1.1$ and the upper bound on norm of the control input is $U_{max} = 0.5$. For this problem three areas are consider. Area 1 with weight W_1 includes the initial point and thus contains the first phase of the trajectory. Area 2 is a small square with weight W_2 that includes the 2nd phase of the

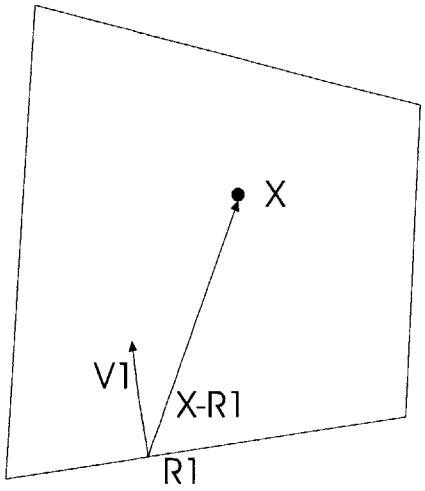


Figure 4.6: Polygon with normal vector $\boldsymbol{v}_1$ and point $\boldsymbol{r}_1$ in one side. Since $\boldsymbol{x}$ satisfies ${\boldsymbol{v}_1}^T [\boldsymbol{x} - \boldsymbol{r}_1] \geq 0$ for all faces, it must be inside

trajectory. Area 3 with weight W_3 contains the final point and thus the last phase of the trajectory. The shapes of the areas and the initial guess are shown in Figure 4.7. This initial guess was used to predict the phases and assign the constraints to each phase. The algorithms were tested with different combinations of weights.

Table 4.2: Results for weighted path problem

Software	Cost (T)	Computation time (secs)	Iterations (fn calls)
DIDO1	43.2	48	27289
NTG1	43.4	4	3960
DIDO2	16.5	26	14087
NTG2	16.5	4	3255

Figure 4.8 shows the solution using DIDO with a surface weight of $W_{out} = 1.5$ and weight of the central rectangle of $W_{in} = 0.3$ (Example 1). The solution was found in 48 seconds after 27289 cost function calls, with a cost of 43.2. This figure shows that instead of generating a straight line trajectory, the trajectory is longer inside the box and minimizes the distance outside of the box where the weight is larger. Figure 4.9 shows the opposite case (Example 2). In this figure the path length inside the box is minimal. Similarly, Figure 4.10 shows the solution found with NTG for weights $W_{out} = 1.5$ and $W_{in} = 0.3$ after 4 seconds and 3960 iterations (cost function calls), with a cost of 43.4. Figure 4.10 shows the trajectory with the inverted weights of

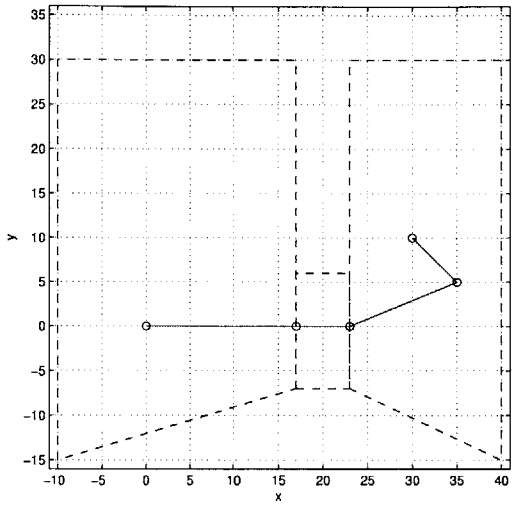


Figure 4.7: Areas 1 to 3 and initial guess with nodes

$W_{out} = 0.3$ and $W_{in} = 1.5$, found after 4 seconds and 3255 cost function calls, with a cost of 16.5.

The results are summarized in Table 4.2, and they are similar to those of the sensor problem in section 4.6. The NTG runs are consistently faster than those of DIDO, due to a combination of less iterations and faster execution of the compiled code, while the costs are very similar. Figures 4.8 to 4.11 have some similarities to the figures for the sensor problem in section 4.6. The trajectories are mostly smooth, other than perhaps at the transition between regions, and the distribution of the nodes show a pattern similar to the trajectories for the sensor problem. The DIDO trajectories of Figures 4.8 and 4.9 are based on the initial guess shown in Figure 4.7. The final trajectories are clearly different from the initial guess, since they are smooth and have a better cost, but they are not far. This result can be expected, because the initial guess is used to determine the number of phases in the problem and assign the constraints to each phase, so the final trajectories are constrained to go over the same areas as the initial guess. Figures 4.8 to 4.11 show the advantages of using a nonlinear optimal control tool in addition to an initial guess, because the shapes of the trajectories have been adjusted to minimize the cost. In the cases where the small area has less weight (Figures 4.8 and 4.10 for DIDO and NTG respectively), the trajectories maximize the length inside the small area. Figures 4.9 and 4.11 show

the opposite case, when the small area has the larger weight, in which the trajectory inside the small area is minimized.

4.9 Summary

This chapter presents how to pose a nonlinear optimal control problem with discontinuous constraints as a multi-phase problem. This chapter also shows how to transform this multi-phase problem into a continuous nonlinear optimal control problem of the type solvable by direct transcription methods such as DIDO and NTG. The procedure is then illustrated with two different discontinuous problems, the UAV sensor problem, and the weighted minimum path problem. The performance and the output of DIDO and NTG are shown for these problems. Using this method and with a reasonable initial guess, a discontinuous nonlinear control problem can be solved with DIDO or NTG. The examples also show that DIDO is much slower than NTG, largely because it runs in MATLAB, while NTG is compiled from C.

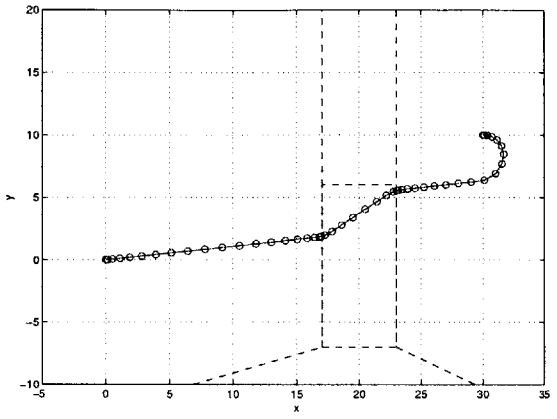


Figure 4.8: DIDO trajectory with weight 0.3 for small box and 1.5 for the rest. Solver tries to maximize length inside small area. Ratio of weights is proportional to ratio of angles

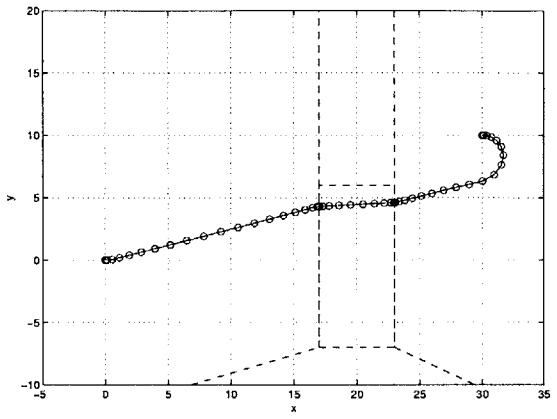


Figure 4.9: DIDO trajectory with weight 1.5 for small box and 0.3 for the rest. Solver tries to minimize length inside small area

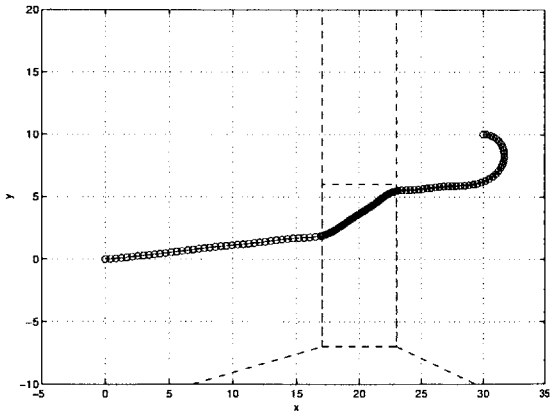


Figure 4.10: NTG trajectory with weight 0.3 for small box and 1.5 for the rest

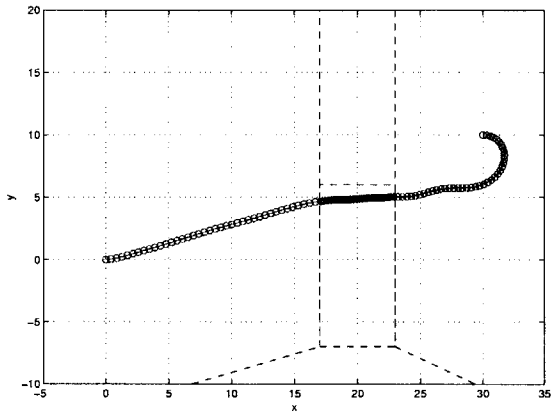


Figure 4.11: NTG trajectory with weight 1.5 for small box and 0.3 for the rest

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Designing spacecraft reconfiguration maneuvers is challenging because it includes non-linear attitude dynamics, difficult non-convex constraints, and high dimensionality ($6N$ DOF) due to coupling of the multiple spacecraft states in the constraints. Chapter 2 presents a method that can solve for reconfigurations of up to 16 spacecraft. The essential feature of this method is the separation into a simplified path planning problem without differential constraints to obtain a feasible solution, which is then improved by a smoothing operation. The first step is solved using Rapidly-exploring Random Trees [24]. The second step consists of an optimization by iteratively solving a linear program using a linearization of the cost function, dynamics, and constraints about the initial feasible solution.

Chapter 3 introduces two extensions to the basic Rapidly-exploring Random Tree algorithm that greatly decrease its computation time. These two extensions are based on the notion that the basic RRT is very good at searching for a trajectory at the global scale, but that it can be improved at the local scale. The extensions consist of replacing the function that connects two points in the basic RRT, which is a simple direct connection, by functions that use some local information about the constraints. These new functions are better at connecting two points in difficult problems, and as a result they improve the overall performance of the whole RRT. One of the functions

is based on probing the local free space with a number of random trials, and the other uses artificial potential functions.

These algorithms and the basic RRT are illustrated and compared using a basic 2D example. The comparison shows that for a simple example the new connection functions find the solution with less iterations than the original function, but not faster. However, for a difficult example the new connection functions perform much better than the direct connection in terms of reliability and computation time. The basic RRT is also compared to the potential connection function by solving the spacecraft reconfiguration problem, and the new RRT is shown to be more reliable and 100 times faster. The new *planner* is then demonstrated by solving large problems involving up to 16 spacecraft (96 DOF) and many constraints.

The spacecraft reconfiguration problem can also be solved by a nonlinear optimal control tool. In Chapter 2 the software DIDO is used for this purpose and its performance is compared with the *planner/smoother* method. The results show that DIDO does not seem to take advantage of a feasible initial guess, with the exception of some constrained problems, and in cases where the initial guess was not smooth DIDO performs better without it. For problems with more than one spacecraft DIDO failed to find any solutions. For simple problems with only one spacecraft and no more than two constraints, DIDO found solutions with a slightly better cost than the solutions by the *planner/smoother*, but DIDO was much slower and less reliable.

Finally, Chapter 4 presents how to pose a nonlinear optimal control problem with discontinuous constraints as a multi-phase problem. This chapter also shows how to transform this multi-phase problem into a continuous nonlinear optimal control problem of the type solvable by direct transcription methods such as DIDO and NTG. The procedure is then illustrated with two different discontinuous problems, the UAV sensor problem, and the weighted minimum path problem. The performance and the output of DIDO and NTG are shown for these problems. Using this method and with a reasonable initial guess, a discontinuous nonlinear control problem can be solved with DIDO or NTG.

5.2 Future Work

The work presented here can be extended in two possible directions. One direction is to improve the spacecraft reconfiguration algorithm. It would useful to try a different feasible nonlinear solver for the *smoother*, perhaps based on sequential quadratic programming, such as CFSQP [26], combined with a formulation based on a recent direct transcription method such as the Pseudospectral Legendre method on which DIDO is based. A possible continuation of the work in the new connection functions would be to compare more thoroughly the potential and the random connection function, perhaps with respect to the spacecraft reconfiguration problem. The potential function method in particular can be tried with a different feasible nonlinear optimizer as well, or even with semidefinite programming which has been proposed as an effective potential-function like solution to the attitude control problem [22]. The other direction is to study further the new connection functions shown here to explain why the slow connection function performs better, and to find the best trade-off between fast and simple, versus slow and effective.

Appendix A

UAV 2D path planning with DIDO and NTG

The minimum time UAV path planning problem with obstacles is a useful problem that is simple enough that can be use to illustrate the implementation of path planning problems with DIDO and NTG. In this problem the dynamics are just a double integrator in each dimension with upper bounds in the velocity and control input, plus a minimum velocity bound to account for the minimum turning radius.

A.1 The UAV Problem with Obstacles

Consider the optimal control problem, similar to the problem described in section 4.2. Minimize the cost functional

$$J = \int_O^T dt \tag{A.1}$$

subject to the dynamic constraints

$$\frac{d\boldsymbol{x}_1}{dt} = \boldsymbol{x}_2(t) \tag{A.2}$$

$$\frac{d\boldsymbol{x}_2}{dt} = \boldsymbol{u}(t) \tag{A.3}$$

which are bounded as

$$V_{min} \leq \|\boldsymbol{x}_2(t)\| \leq V_{max} \tag{A.4}$$

$$\|\boldsymbol{u}(t)\| \leq U_{max} \tag{A.5}$$

and the initial and final constraints for position and velocity

$$\boldsymbol{x}_1(0) = \boldsymbol{x}_{1,0} \tag{A.6}$$

$$\boldsymbol{x}_1(T) = \boldsymbol{x}_{1,f} \tag{A.7}$$

$$\boldsymbol{x}_2(0) = \boldsymbol{x}_{2,0} \tag{A.8}$$

$$\boldsymbol{x}_2(T) = \boldsymbol{x}_{2,f} \tag{A.9}$$

where $\boldsymbol{x}_1(t) \in \mathbb{R}^2$ is the position, $\boldsymbol{x}_2(t) \in \mathbb{R}^2$ is the velocity $\boldsymbol{u}(t) \in \mathbb{R}^2$ is the control input, $t \in \mathbb{R}$ is the independent time. Additionally, the circular obstacles are described by the path constraints

$$R_i \leq \|\boldsymbol{x}_1(t) - \boldsymbol{r}_i\|, \forall i \tag{A.10}$$

where $i \in 1, \ldots, m$ the number of obstacles, $\boldsymbol{r}_i$ and R_i are the position and radius of obstacle i

A.2 Implementation with DIDO

Figure A.1 shows the trajectory generated by DIDO for a problem with initial position $\boldsymbol{x}_{1,0} = [0, 0]^T$ and velocity $\boldsymbol{x}_{2,0} = [1, 0]^T$, and final position $\boldsymbol{x}_{1,f} = [30, 10]^T$ and velocity $\boldsymbol{x}_{2,f} = [-1, 0]^T$. The example includes two obstacles centered at $\boldsymbol{r}_1 = [10, 0]^T$ and $\boldsymbol{r}_2 = [25, 4]^T$ with radius $R_1 = 2$ and $R_2 = 3$ respectively.

This trajectory of 35 nodes was generated 28.4 seconds and 18440 iterations (cost function calls), for a cost of 35.5 (time in seconds).

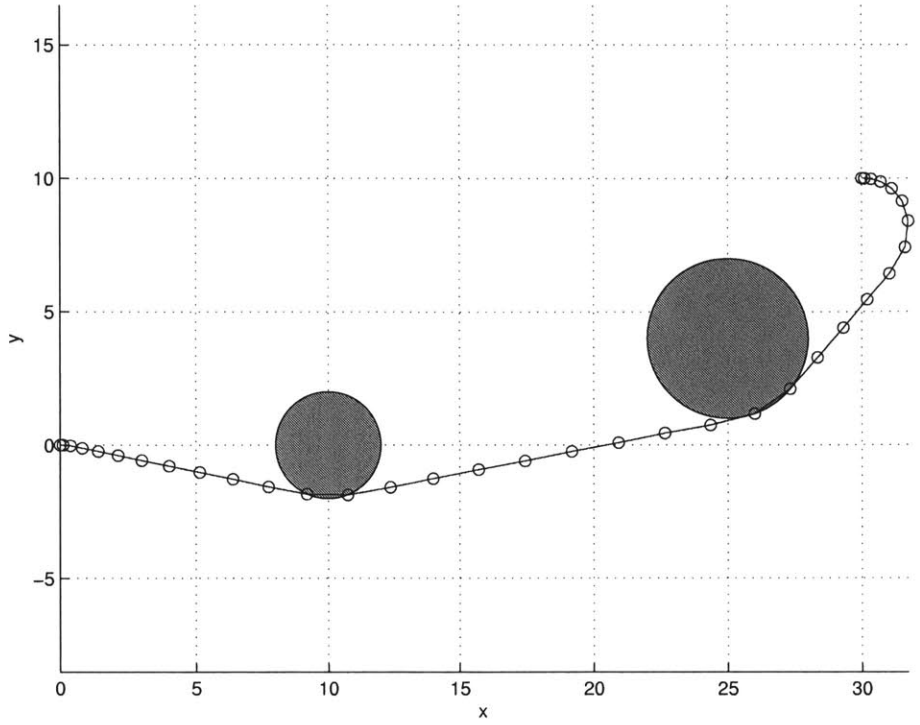


Figure A.1: Trajectory generated by DIDO for the 2D UAV problem

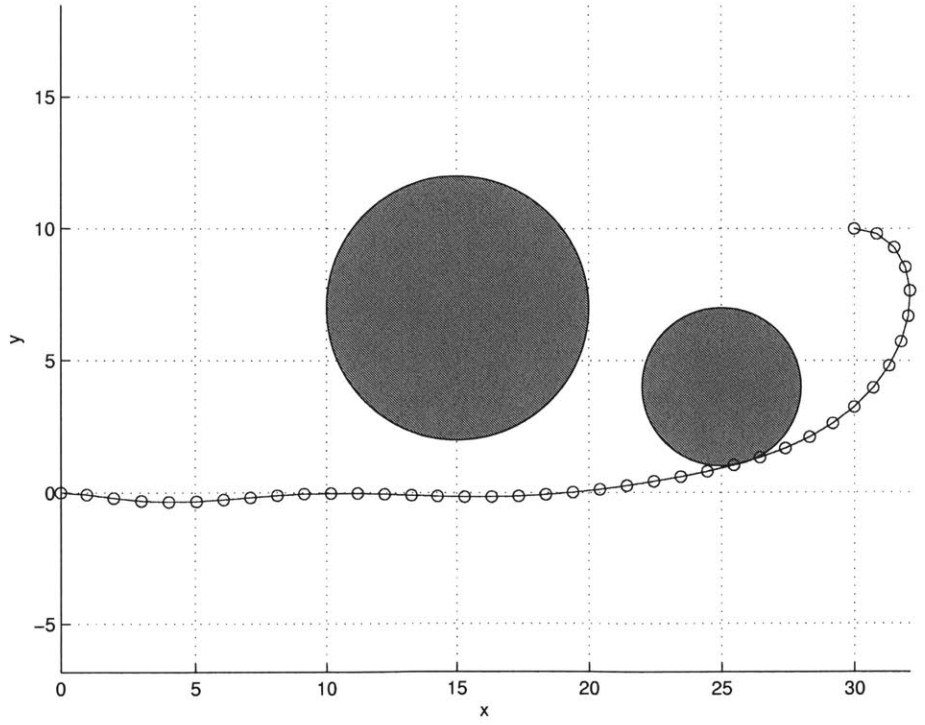


Figure A.2: Trajectory generated by NTG for the 2D UAV problem

A.3 Implementation with NTG

Figure A.2 shows the trajectory generated by NTG for a problem with initial position $\boldsymbol{x}_{1,0} = [0, 0]^T$ and velocity $\boldsymbol{x}_{2,0} = [1, 0]^T$, and final position $\boldsymbol{x}_{1,f} = [30, 10]^T$ and velocity $\boldsymbol{x}_{2,f} = [-1, 0]^T$. The example includes two obstacles centered at $\boldsymbol{r}_1 = [15, 7]^T$ and $\boldsymbol{r}_2 = [25, 4]^T$ with radius $R_1 = 5$ and $R_2 = 3$ respectively.

This trajectory was generated after less than 1 second and 1080 iterations (cost function calls), for a cost of 36.45 (time in seconds).

Bibliography

[1] C. Beichman, N. Woolf, and C. Lindensmith, eds., "The Terrestrial Planet Finder (TPF):A NASA Origins Program to Search for Habitable Planets," JPL Pub. 99–3, Cal. Inst. Tech., Pasadena, CA, 1999.

[2] R. Bellman, "Adaptive Control Processes: A Guided Tour," Princeton University Press, 1961.

[3] J. T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control, and Dynamics,* Vol. 21, No. 2, March-April 1998.

[4] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," In Proceedings of the IEEE *International Conference on Robotics and Automation,* pp. 1018–1023, 1999.

[5] J. F. Canny, "The Complexity of Robot Motion Planning," MIT Press, 1988.

[6] Carpenter et al., "The Stellar Imager (SI): A Revolutionary Large-Baseline Imaging Interferometer at the Sun-Earth L2 Point," in *Proceedings of SPIE meeting in Glasgow, Scotland,* June 2004

[7] C. M. Clark, "Dynamic Robot Networks: A Coordination Platform for Multi-Robot Systems", PhD Thesis, June 2004.

[8] J. Elnagar, M. A. Kazemi, and M. Razzaghi, "The Pseudospectral Legendre Method for Discretizing Optimal Control Problems," *IEEE Transactions on Automatic Control,* Vol. 40, No. 10, 1995, pp. 1793-1796.

[9] R. Fletcher, "Practical Methods of Optimization," John Wiley and Sons, 1987.

[10] E. Frazzoli, M. Dahleh, E. Feron, R. Kornfeld, "A Randomized Attitude Slew Planning Algorithm For Autonomous Spacecraft," *AIAA Guidance, Navigation and Control Conf.*, AIAA 2001–4155.

[11] E. Frazzoli, "Quasi-Random Algorithms for Real-Time Spacecraft Motion Planning and Coordination" *International Astronautical Congress*, Houston, TX, 2002.

[12] S. Ge, and Y. Cui, "New potential functions for mobile robot path planning," in IEEE Transactions on *Robotics and Auto.*, Vol. 16, no. 5, pp. 615–620, 2000.

[13] P. E. Gill, W. Murray, M. A. Saunders, and M. Wright. "User's Guide for NPSOL 5.0: A Fortran Package for Nonlinear Programming." Systems Opti- mization Laboratory, Stanford University, Stanford, CA 94305.

[14] P. E. Gill, W. Murray, M. A. Saunders, and M. Wright. "User's Guide to SNOPT 5.3: A Fortran Package for Large-Scale Nonlinear Programming," December 1998.

[15] H. Hablani, "Attitude Commands Avoiding Bright Objects and Maintaining Communication with Ground Station," AIAA *Journal of Guidance, Control, and Dynamics,* Vol. 22, No. 6, pp. 759–767.

[16] C. Hargraves, and S. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *AIAA J. Guidance and Control,* 10:338–342, 1987.

[17] D. Hsu, T. Jian, J. Reif, and Z. Sun, "The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners," In Proceedings of the IEEE *International Conference on Robotics and Automation,* Taipei, 2003.

[18] J. Junkins, and J. D. Turner, "Optimal Spacecraft Rotational Maneuvers," Elsevier, 1986.

[19] L. E. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," IEEE *Trans on Robotics and Auto.,* (12)4, pp. 566–580, 1996.

[20] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robotics Research*, (5), no. 1, pp. 90–98, 1986.

[21] J. Kim, and J. Hespanha, "Discrete Approximations to Continuous Shortest-Path: Application to Minimum-Risk Path Planning for Groups of UAVs," In *Proc. of the 42nd Conf. on Decision and Contr.*, December 2003.

[22] Y. Kim, and M. Mesbahi, "Quadratically constrained attitude control via semi-definite programming," *IEEE Transactions on Automatic Control* 49 (5): 731–735, 2004.

[23] J. C. Latombe, "Robot Motion Planning", Kluwer Academic Publishers, Boston, MA, 1991.

[24] S. M. LaValle, and J. J. Kuffner, "Randomized Kinodynamic Planning," *International Journal of Robotics Research,* Vol. 20, No. 5, pp. 378–400, May 2001.

[25] S. M. LaValle, "Planning Algorithms," [Online] http://msl.cs.uiuc.edu /planning/, 2005.

[26] C. Lawrance, J. Zhou, and A. Tits. "User's guide for CFSQP Version 2.5." Institute for Systems Research, University of Maryland, College Park, MD 20742.

[27] C.T. Lawrence, and A.L. Tits, "A Computationally Efficient Feasible Sequential Quadratic Programming Algorithm," SIAM J. Optimization, Vol. 11, No. 4, pp. 1092–1118, 2001.

[28] P. Lawson, "The Terrestrial Planet Finder," *IEEE Aero. Conf.*, pp. 2005-2011, 2001.

[29] J. Leitner, F. Bauer, D. Folta, M. Moreau, R. Carpenter, and J. How, "Distributed Spacecraft Systems Develop New GPS Capabilities," in *GPS World: Formation Flight in Space* Feb. 2002.

[30] S. R. Lindemann, and S. M. LaValle, "Incremental low-discrepancy lattice methods for motion planning," *Proc. IEEE Intl. Conf. on Robotics and Automation,* 2003.

[31] C. R. McInnes, "Large Angle Slew Maneuvers with Autonomous Sun Vector Avoidance," AIAA *JGCD,* Vol. 17, No. 4, pp. 875–877.

[32] M. B. Milam, K. Mushambi, and R. M. Murray, "A new computational approach to real-time trajectory generation for constrained mechanical systems," in *Proc. IEEE Conf. Decision and Control,* Sydney, Australia, Dec. 2000, pp. 845-851.

[33] M. B. Milam, "Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems," PhD Thesis, Caltech, 2003.

[34] NASA, "Human And Robotic Technology formulation plan–July 2004," [Online], http:// exploration.nasa.gov/ documents/ nova_29july04b.pdf, July 2004.

[35] NASA, *The Stellar Imager mission,* http:// hires.gsfc.nasa.gov/ -si/

[36] NASA, *The Terrestrial Planet Finder mission,* http:// planetquest.jpl.nasa.gov/ TPF/

[37] N. Petit, M. B. Milam, and R. M. Murray, "Inversion based constrained trajectory optimization," in *Proc. 2001 IFAC Symp. Nonlinear Control Systems and Design (NOLCOS),* 2001.

[38] J. Phillips, L. E. Kavraki, and N. Bedrosian , "Probabilistic Optimization Applied to Spacecraft Rendezvous and Docking," In *13th American Astronomical Society/AIAA - Space Flight Mechanics Meeting,* Puerto Rico, February 2003.

[39] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, "The Mathematical Theory of Optimal Processes," Wiley-Interscience, 1962.

[40] A. Rao, "Extension of a Pseudospectral Legendre Method to Non-Sequential Multiple-Phase Optimal Control Problems," *AIAA Guidance, Navigation, and Control Conference and Exhibit,* Austin, Texas, August, 2003.

[41] J. H. Reif, "Complexity of the Mover's Problem and Generalizations," *20th Annual IEEE Symposium on Foundations of Computer Science,* San Juan, Puerto Rico, October 1979, pp. 421–427.

[42] A. Richards, T. Schouwenaars, J. P. How, and E. Feron, "Spacecraft Trajectory Planning With Collision and Plume Avoidance Using Mixed-Integer Linear Programming," AIAA *Journal of Guidance, Control, and Dynamics,* Vol. 25, No. 4, pp. 755–765, Aug. 2002.

[43] E. Rimon, and D. E. Koditschek, "Exact robot navigation using artificial potential functions," in IEEE Transactions on *Robotics and Automation*, Vol. 8, No. 5, pp. 501–518, Oct. 1992.

[44] I. M. Ross, and F. Fahroo, "Legendre Pseudospectral Approximations of Optimal Control Problems," *Lecture Notes in Control and Information Sciences*, New York: Springer-Verlag, 2003, vol. 295, pp. 327-342.

[45] I. M. Ross, and F. Fahroo, "Pseudospectral Knotting Methods for Solving Optimal Control Problems," In *AIAA Journal of Guidance, Control and Dynamics,* Vol. 27, No. 3, MayJune 2004.

[46] I. M. Ross, and F. Fahroo, "Users Manual for DIDO 2002: A MATLAB Application Package for Dynamic Optimization, Dept. of Aeronautics and Astronautics, NPS Technical Rept. AA-02-002, Naval Postgraduate School, Monterey, CA, June 2002.

[47] G. Sánchez, and J. C. Latombe, "On Delaying Collision Checking in PRM Planning - Application to Multi-Robot Coordination," In *Int. J. of Robotics Research,* Vol. 21, No. 1, pp. 5–26, January 2002.

[48] D. P. Scharf, and S. R. Ploen, F. Y. Hadaegh, J. A. Keim, and L. H. Phan, "Guaranteed Initialization of Distributed Spacecraft Formations," *AIAA Guidance, Navigation and Control Conference*, AIAA 2003–5590, August 2003.

[49] J. Shoemaker, "Orbital Express Space Operations Architecture / ASTRO," http://www.darpa.mil/tto/programs/oe.html, DARPA, Apr 2001.

[50] O. von Stryk, and R. Burlisch, "Direct and Indirect Methods for Trajectory Optimization," *Annals of Operations Research,* 37:357–373, 1992.

[51] C. Sultan, S. Seereeram, R. Mehra, and F. Hadaegh, "Energy Optimal Reconfiguration for Large-Scale Formation Flying," In *Proceedings of the American Control Conference,* pp. 2986–2991, July 2004.

[52] Z. Sun, and J.H. Reif, "On Finding Approximate Optimal Paths in Weighted Regions," *Journal of Algorithms,* Volume 56, 2005.

[53] S. J. Wright, and M. J. Tenny, "A Feasible Trust-Region Sequential Quadratic Programming Algorithm," SIAM Journal on Optimization 14, pp. 1074–1105, 2004.